**VŠB – Technická univerzita Ostrava**
**Fakulta elektrotechniky a informatiky**
**Katedra telekomunikační techniky**

# Rozšíření Vektorového animátoru - grafická část
# Extension of Vector Animator - Graphical Part

**2013** **Martin Gold**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Martin Gold**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Rozšíření Vektorového animátoru - grafická část
Extension of Vector Animator - Graphical Part

Zásady pro vypracování:

Cílem práce je rozšířit stávající projekt (jazyk Java) Vektorového animátoru o rozsáhlejší možnosti vkládání objektů (lichoběžník, hvězda, ...), jejich seskupování a editaci vlastností (výplň, obrysy, typ výplně průhlednost). Další částí je vylepšení grafického uživatelského rozhraní.
Program bude rozšířen o:
1. Možnosti vkládání dalších objektů (ovál, obdélník s kulatými rohy, hvězda, lichoběžník, n-úhelník, kvádr, šipka, ...).
2. Editovat vlastnosti objektů (barva, výplň, obrys, ...).
3. Možnost transformace objektů (změna velikosti, zešikmení).
4. Zobrazení stromové struktury vložených objektů.
Práce musí obsahovat:
1. Popis řešení pomocí jazyka UML.
2. Implementaci popsané funkcionality.
3. Programátorskou a uživatelskou dokumentaci.

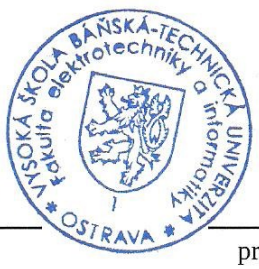
Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013




doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

## Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 2. 5. 2013

.....
podpis studenta

# Poděkování

Rád bych poděkoval Ing. Davidu Ježkovi, Ph.D. za odbornou pomoc a konzultaci při vytváření této bakalářské práce a za vstřícnost.

# Abstrakt

Hlavním cílem této práce je rozšíření aplikace pro tvorbu animací o pokročilejší funkce. Rozšíření by mělo hlavně obsahovat nové možnosti vkládání objektů, podporu pro následnou editaci jejich vlastností a seskupování do větších celků. Dalším bodem práce je výrazně vylepšit grafické uživatelské rozhraní, k dosažení lepší orientace v aplikaci. Základní konstrukce tohoto softwaru již byla vypracována jako diplomová práce studenta Karla Šuty. Výsledná aplikace by měla sloužit k vytváření animací pro podporu výuky.



# Klíčová slova

Java, aplikace, rozšíření, GUI, tvary, grafika, vektor, editace, animace, panel, seznam

# Abstract

The main goal of this work is to extend application for creating of animations of advanced functions. Extension should mainly contain new options of inserting objects, support for their subsequent editing their properties and grouping into larger unit. Next point of this work is significantly upgrade graphical user interface, to reach better orientation in application. Basic construction of this software already been created as disertation by student called Karel Suta. Resulting application should serve to creating animation to support education.



# Key words

Java, application, extension, GUI, shapes, graphics, vector, editating, animation, panel, list

# Seznam použitých zkratek

| **Zkratka** | Anglický význam | Český význam |
|---|---|---|
| **API** | Application Programming Interface | Aplikační programové rozhraní |
| **2D** | Two-dimensional | Dvourozměrný |
| **GUI** | Graphical User Interface | Grafické uživatelské rozhraní |
| **Barevná schémata** | | |
| **HSV** | Hue, saturation, value | Odstín, sytost, jas |
| **HSL** | Hue, saturation, lightness | Odstín, sytost, světlost |
| **RGB** | Red, green, blue | Červená, zelená, modrá |
| **CMYK** | Cyan, magenta, yellow, key | Azurová, purpurová, žlutá, klíčová (černá) |

# Obsah

# 1 Úvod

Tato bakalářská práce se zabývá rozšířením již vytvořeného prototypu nástroje pro práci s vektorovou grafikou. Aplikace byla rozšířena o vkládání nových typů objektů, které by mohly být potřebné při jejím použití ve výuce. S tím je spojeno i rozšíření možnosti práce s těmito objekty. Detailní popis původního prototypu aplikace lze nalézt v [7].

V první části práce byly zanalyzovány potřebné vlastnosti pro tento program. Snažil jsem se zjistit, které vlastnosti jsou u podobných již existujících nástrojů používány. Ohled jsem bral však i na to, jaké možnosti by byly při tvorbě výukových animací důležité a jaké by byly už naopak zbytečné.

V druhé části práce jsem se zaměřil na popis použitých technologií. V této aplikaci byly totiž použity mnohé třídy a knihovny, které nejsou v širším povědomí moc známé, nebo nejsou tak často využívané. Proto jsem se rozhodl alespoň nastínit jejich funkci, pro lepší orientaci v kódu.

V další části jsem vypsal základní nedostatky, které byly v původní aplikaci obsaženy. Tyto problémy způsobily velké komplikace při vypracovávání této bakalářské práce. Díky velké provázanosti jednotlivých tříd totiž bylo potřeba přepsat nebo upravit podstatnou část aplikace.

V následující části jsem se zaměřil na popis nového systému vykreslování tvarů. Nejprve je zde popsán obecný algoritmus společný pro většinu objektů a následně jsou v příslušných kapitolách popsány konkrétní specifikace pro každý typ tvaru.

V šesté části této práce je popsána implementace modifikačních dialogů, které uživateli umožňují editovat vybrané vlastnosti tvaru. Obsažen je zde i podrobnější popis možností každého typu dialogu.

Sedmá část je následně věnována popisu jednotlivých typů komponent použitých k sestavení modifikačního dialogu. Tento nový přístup přinesl velké zjednodušení kódu a lehčí přístup k případné implementaci nového modifikačního okna.

Následující část této práce je zaměřena na popsání implementace seznamu všech přítomných objektů na kreslícím plátně. Stručně jsou zde popsány i základní třídy, které byly k tomuto účelu využity.

Poslední část se zabývá řešením nedostatků, které byly zmíněny ve čtvrté kapitole této práce. Kromě jejich analýzy je zde i popsán postup, který byl použit pro jejich odstranění.

# 2 Analýza vlastností dostupných vektorových nástrojů

V této části bych se chtěl věnovat srovnáním aplikací dostupných na trhu. Primárně bych se ale chtěl zaměřit na možnosti vykreslování objektů a změny jejich vlastností. Je zřejmé, že mnou vyvíjená aplikace nebude mít právo srovnávat se s komerčními produkty, jako jsou například Adobe Illustrator od firmy Adobe Systems nebo třeba CorelDRAW z dílny firmy Corel Corporation. Mým cílem je ovšem vytvořit aplikaci, která bude sloužit jako výuková pomůcka a tudíž zde není nezbytně nutné implementovat složité funkce, které by nakonec stejně nebyly využívány. Na druhou stranu by ale tato aplikace měla splňovat obecné principy, které jsou v této oblasti běžné.

## 2.1 Microsoft PowerPoint 2010

Tento program se stal velkým zdrojem inspirace pro vypracování této bakalářské práce. I když tento nástroj nabízí velkou škálu možností, mým cílem bylo zaměřit se na tvorbu a následnou modifikaci grafických objektů. V tomto ohledu jsem dospěl k mnoha kladům, ale také k některým záporům.

Kladně bych například hodnotil jednoduchost při vytváření nových objektů. Uživateli je nabízena opravdu velká škála předdefinovaných tvarů, ze kterých si jde poměrně lehce vybrat a vytvořit tak požadovaný výsledek. Také bych hodnotil kladně i náhled při tvoření objektu, který v původním prototypu mnou rozšiřovaného animačního nástroje chyběl. Uživateli totiž tato základní vlastnost velmi usnadní práci. Dalším kladem je dle mého názoru opravdu jednoduchá manipulace s rozměry objektu, kdy stačí označit a přemístit řídící bod objektu a přepočítá se tím celá jeho geometrie do požadovaného stavu. Další užitečnou funkcí je přeuspořádání pořadí vykreslování tvarů. V PowerPointu je však tato možnost podle mého názoru zařazena jen jako doplňková. Moje představa však je, že by měla být prosazovanější, poněvadž si myslím, že může být poměrně často vyžadována. Jako velký nadstandard bych nakonec hodnotil funkci pro úpravu bodů, kdy může uživatel upravit jakýkoliv bod, ze kterých je tvar sestaven, a úplně tak změnit jeho celkový vzhled.

Záporně bych především hodnotil uživatelskou přístupnost modifikací jednotlivých objektů. Mám na mysli například změnu výplně, nebo zvolení stylu ohraničení. Zde je ale nutno podotknout, že PowerPoint není nástroj primárně určený na vytváření a modifikaci vektorových obrazců, takže lze tento nedostatek pochopit. U mé aplikace bych však chtěl tento zápor odstranit tím, že uživateli okamžitě nabídnu nabídku všech měnných parametrů v bočním panelu. Jako další nedostatek bych zde ještě uvedl například špatně vyřešené označování objektů pomocí výběrového obdélníku (tahem myši). Jde totiž podle mého názoru o naprosto esenciální funkčnost. Nicméně původní verze tohoto vektorového animátoru již má tuto možnost naimplementovanou správně, takže se tímto problémem už nemusím dále zaobírat.

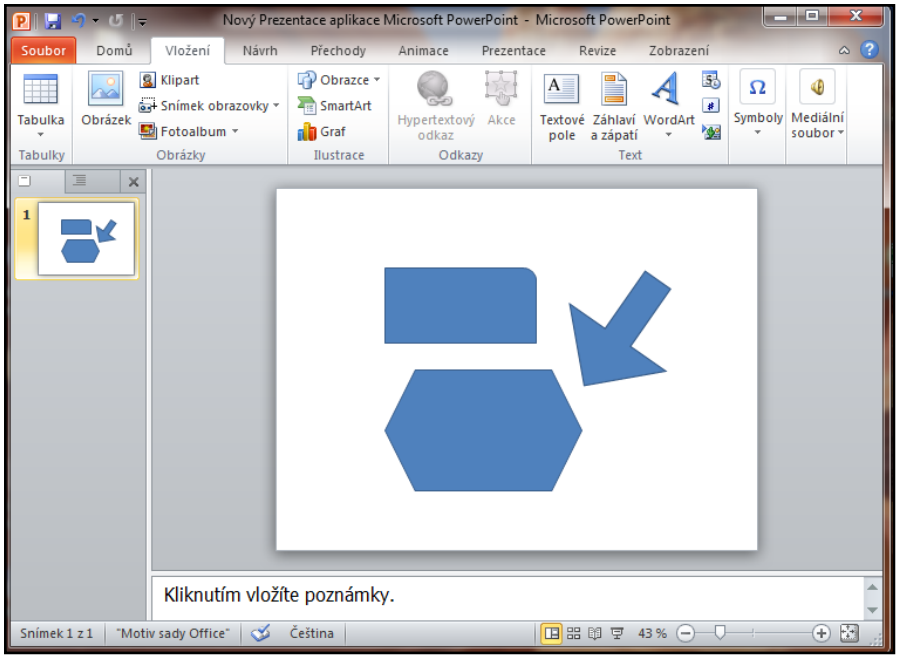
## 2.2 EVE – Embedded Vector Editor 3.56

Jako další nástroj pro analýzu vlastností jsem si vybral program EVE, který na rozdíl od výše jmenovaného PowerPointu 2010, je poskytován jako freeware a je určen pouze pro práci s vektorovou grafikou. Tento nástroj je velmi jednoduchý, ale přesto nabízí poměrně velkou škálu možností.

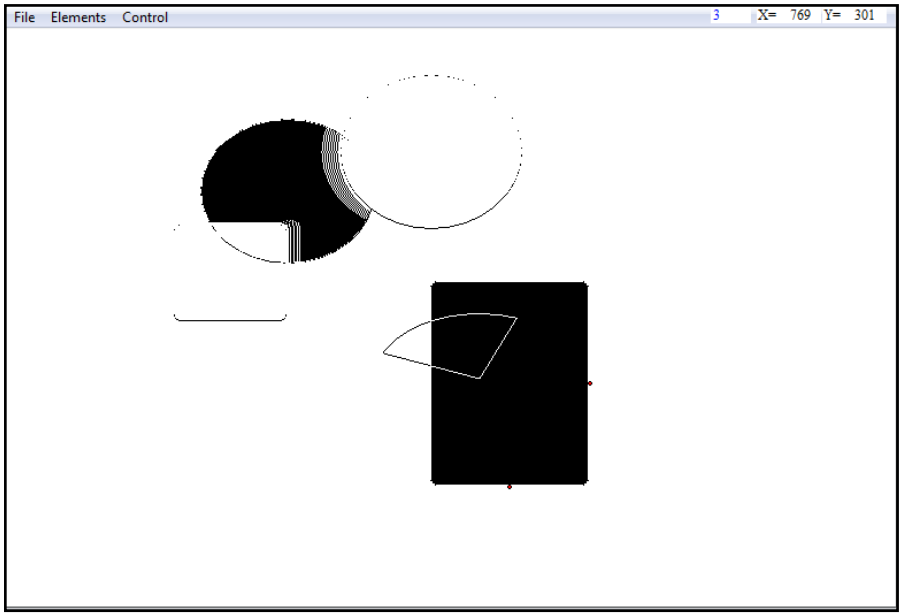
Kladně bych například hodnotil jednoduchost programu, který se zaměřuje čistě na práci s vektorovými obrazci. Pozitivní je například i to, že program nabízí uživateli výčet základních tvarů, ze kterých si může uživatel vybrat. Dále bych ještě ocenil možnost jednoduchých úprav tvarů pomocí přesouvání řídících bodů.

Záporně bych naopak hodnotil zpracování grafického uživatelského rozhraní. K dispozici je z počátku uživateli nabídnuta jen pracovní plocha a řádek nabídek (menu bar) pouze se třemi možnostmi. Uživatel tak musí pro každou akci otevírat potřebnou nabídku a vybírat ze seznamu možností, což mi přijde velmi špatně vyřešené. Jak další zápor bych viděl špatné překreslování pracovní plochy. Objekty totiž často problikávají nebo nechávají při přesunu na plátně pozůstatky objektů (takzvané „duchy"). Dále bych snad ještě uvedl chybějící náhled při tvoření objektu, nebo možnost rotace objektu pouze o násobky devadesáti stupňů.

Tento program sice obsahuje velké množství chyb a nedostatků, ale právě proto byla pro mě jeho analýza velmi užitečná. Získal jsem alespoň povědomí o možných záporech a mohl jsem se jim při mé implementaci snažit co nejvíce vyhýbat.




*Obrázek 2 .1: Ukázka grafického uživatelského rozhraní aplikace Microsoft PowerPoint 2010*




*Obrázek 2 .2: Ukázka uživatelského rozhraní aplikace EVE-Embedded Vector Editor 3.56*

# 3 Použité technologie

V průběhu vypracovávání bakalářské práce byly použity třídy a balíčky, jejichž znalost není příliš rozšířená, proto bych zde chtěl vypsat alespoň nejdůležitější z nich a okrajově nastínit jejich možnosti a funkce.

## 3.1 Graphics2D

Tato třída je základní třídou pro renderování 2D tvarů textu a obrázků na Java platformě. **Graphics2D** dědí z třídy **Graphics**, ale na rozdíl od ní poskytuje sofistikovanější možnosti ovládání geometrie, transformací nebo například správu barev. Více informací můžete nalézt na [1].

## 3.2 AffineTransform

Třída **AffineTransform** je součástí balíčku **java.awt.geom** a poskytuje afinní transformace, které mapují původní 2D souřadnice do transformovaných 2D souřadnic. Poskytovány jsou hlavně metody pro posunutí, otočení, zkosení a změnu měřítka. Kombinací těchto čtyř základních operací jsou v počítačové grafice realizovány veškeré transformace. Hlavním principem je, že každá základní transformace je vyjádřena jako matice o třech řádcích a třech sloupcích (pro transformace ve 2D). Chceme-li složit dvě transformace, vynásobíme v určitém pořadí jejich matice, čímž získáme matici složené transformace. Další informace o této třídě najdete na [1].

## 3.3 Commons BeanUtils

Tato knihovna poskytuje uživatelsky snadný nástroj pro reflexi. V některých případech je totiž velmi komplikované přistupovat ke ***get*** a ***set*** metodám jednotlivých objektů, proto jsem se rozhodl použít tuto knihovnu, abych tento přístup k vlastnostem objektů co nejvíce zjednodušil. Tento přístup k vlastnostem je vidět u vlastně vytvořených komponent pro modifikační panel objektů. Jejich přesné použití je pak dále popsáno v příslušné kapitole této bakalářské práce. Více informací o této knihovně pak můžete nalézt na [3].

# 4 Problémy s předchozím zpracováním

## 4.1 Kreslené objekty

Při přidávání nových typů objektů musel být vyřešen podstatný problém. U kreslených tvarů přítomných v aplikaci byl použit koncept, kdy se každý objekt skládal z úseček, křivek a bodů. Tento model zobrazování však není ideální pro účely této aplikace. Proto jsem se rozhodl více využít možností **API Java2D**. Z tohoto důvodu jsem sice ponechat reprezentaci objektů pomocí řídících bodů, ale v samotném vykreslování tvarů jsem udělal výraznou změnu. Místo zobrazování objektů pomocí úseček a křivek jsem použil předdefinované tvary obsažené v balíčku **java.awt.geom**. Jejich konkrétní použití je popsáno dále v této bakalářské práci u popisu jednotlivých tvarů. Využitím těchto možností jsem dosáhl lepších a rozsáhlejších možností práce s tvary. Velkou výhodou bylo například to, že nově použité objekty byly uzavřené a tudíž u nich bylo možno použít metodu ***fill***, která vytvořila výplň tvaru. Tato metoda přístupu na druhou stranu přinesla větší množství výpočtů pro korektní vykreslení.

## 4.2 Absence náhledu tvořeného objektu

Jedna z prvních věcí, které jsem v aplikaci postrádal, byla možnost náhledu tvořeného objektu. U valné většiny aplikací pro práci s grafikou tato možnost existuje, a tak jsem ji chtěl vytvořit i zde. V původní verzi byla pro vytváření objektů použita třída **ActionMouseCreate**, která ovšem registrovala pouze body stisknutí a uvolnění tlačítka a podle nich pak vytvořila vybraný objekt. Aby však bylo dosaženo požadovaného náhledu při vytváření tvaru, vytvořil jsem třídu **ActionMouseDraggCreate**. Popis její implementace je uveden dále v této bakalářské práci.

## 4.3 Nedostačující vzhled aplikace

Mezi důležité aspekty úspěšného programu patří kromě dobrého naprogramování i přijatelný vzhled samotné aplikace. Ta kromě vzhledové přitažlivosti usnadňuje i orientaci v tomto programu, což bývá u většiny editorů klíčové. Původní aplikace byla napsána spíše jako prototyp se zaměřením na vnitřní architekturu programu. V této bakalářské práci jsem se musel soustředit i na přeuspořádání některých grafických komponent, abych tak dosáhl pro uživatele jednoduššího ovládání.

# 5 Nové objekty

Jak již bylo zmíněno výše v tomto dokumentu, přidávání nových objektů nevyžadovalo pouze implementovat nové tvary na dříve vytvořený systém vykreslování, ale bylo třeba jej celý přepracovat. Původní systém byl totiž postaven na reprezentaci tvarů pomocí jednodušších primitiv, jako jsou úsečky a křivky. Tento způsob je sice jednodušší na vykreslování ale neposkytuje potřebné možnosti, které jsou očekávány od této aplikace. Odstranil jsem proto velkou část toho algoritmu a ponechal jsem v podstatě pouze řídící body, které jsem využil jako základ ke všem výpočtům při vykreslování objektů.

V následujícím oddíle bych se chtěl zaměřit na popis systému vykreslování jednotlivých tvarů. Pro přehlednější popis algoritmu vykreslování, jsem se rozhodl všechny přidané objekty rozdělit do menších podskupin, jejichž systém je stejný nebo alespoň velice podobný. U konkrétních tříd budou následně uvedeny doplňující informace.

## 5.1 Rozhraní a abstraktní třída společná pro všechny tvary

### IShape

Toto rozhraní je společné pro všechny dostupné tvary v této aplikaci. Obsahuje jak základní metody potřebné pro vytvoření nové instance a vykreslení, tak metody pro nastavování nebo zjišťování parametrů jednotlivých objektů. Přístup k těmto hodnotám je důležitý hlavně při animování objektů, poněvadž celý systém animování je založen na práci s obecnými objekty typu **IShape**.

### AbstractShape

**AbstractShape** implementuje rozhraní **IShape** a tvoří abstraktní třídu pro veškerá geometrická primitiva obsažená v této aplikaci. Kvůli snadnějšímu rozšiřování aplikace o nové druhy tvarů bylo co největší množství metod zobecněno a jejich implementace byla přesunuta právě do této abstraktní třídy. Případný programátor přidávající nový tvar objektu do tohoto programu má tak již část algoritmu přednastavenu a musí dopsat pouze konkrétní implementaci metod pro daný objekt. V následující části bude popsána funkcionalita některých klíčových metod obsažených v této abstraktní třídě.

#### *recalculateAngle*

Tato metoda slouží k přepočítání úhlu objektu. Jako vstupní parametry jsou metodě předány indexy dvou bodů objektu. Pro tyto body musí platit, že jestliže je úhel natočení objektu nulový, budou mít tyto dva body stejnou souřadnici y. Postup výpočtu je následující. Nejprve je porovnávaný vektor posunut tak, aby začínal ve středu souřadné soustavy. Pro vypočítání hledaného úhlu je použita funkce arkustangens, její obor hodnot však obsahuje pouze interval od $-\frac{\pi}{2}$ do $\frac{\pi}{2}$, a tak je třeba ještě do tohoto výpočtu zahrnout kvadrant, ve kterém se koncový bod vektoru nachází. Celkový úhel je nakonec realizován ve stupních.

### *drawShadow*

Tato metoda slouží pro vytvoření vrženého stínu objektu. V podstatě jde o sestavení kopie tvaru, která je posunuta o určitý vektor a jako barva výplně je ji přidělena barva stínu. Metoda ve třídě **AbstractShape** je implementována pouze pro tvary vytvořené pomocí **Path2D**. Jelikož neexistuje způsob jak jednoduše posunout celý objekt jako celek, je třeba nejprve pomocí metody ***getPathIterator*** získat všechny vrcholy objektu a připočítat k nim vektor posunutí, jež je definován v parametrech jako odsazení stínu. Z těchto bodů je následně vytvořen nový objekt, jehož výplň je definována určenou barvou stínu. Důležitou vlastností stínu je, že vektor jeho posunutí se nevztahuje k souřadné soustavě objektu, ale počítá se podle souřadné soustavy celého plátna, takže odsazení stínu není ovlivněno úhlem otočení objektu. U některých vložených tvarů je právě kvůli této vlastnosti implementována pozměněná metoda, která tento problém řeší.

### *shearShape*

Tato metoda se stará o správné aplikování modifikace zkosením. Proces této modifikace je ale složitější a zasahuje do více míst algoritmu. Konkrétně v této metodě se provádí pouze výpočet souřadnic řídících bodů, které bude mít zkosený objekt. Nejdříve se celý objekt otočí tak, aby byl jeho výsledný úhel rotace nulový. Poté se provede příprava požadované matice pro modifikaci. U nového souřadného systému je nejprve provedeno zkosení osy y a následně je proveden posun o vektor, jehož velikost závisí na velikosti koeficientu zkosení. Při zkosení totiž dojde k posunutí všech bodů v závislosti na jejich souřadnici y. U této modifikace je ale potřeba, aby souřadnice určených bodů zůstaly pro lepší orientaci zachovány, což zajišťuje právě použitá matice posunutí. Na konci metody se body znovu otočí o požadovaný původní úhel. Nutno ještě dodat, že násobení matic není komutativní operace a tudíž musí být zachováno jejich pořadí. U těchto operací pak platí pravidlo, že násobení transformačních matic se provádí v opačném pořadí, než ve kterém mají být transformace prováděny.

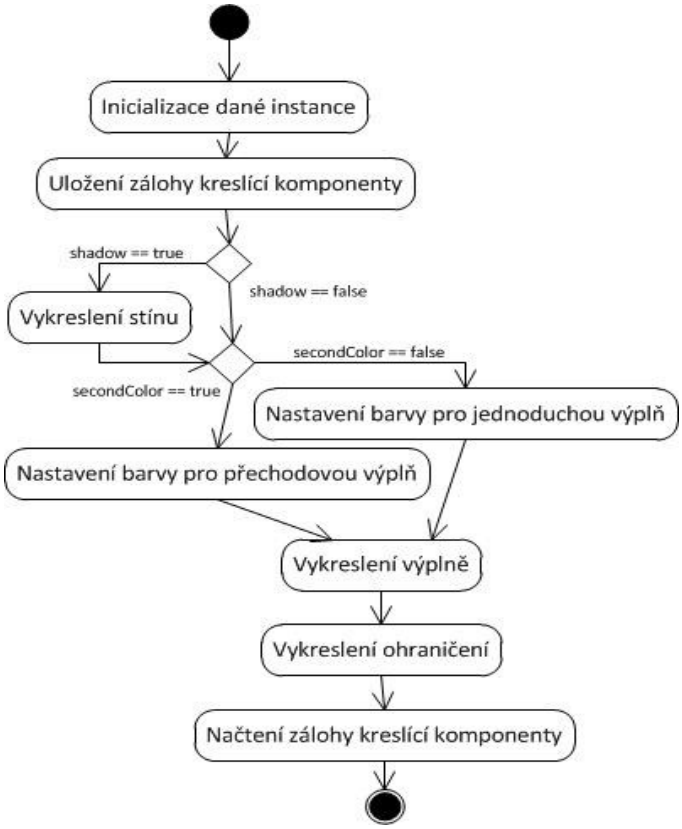
## 5.2 Obecný algoritmus

Jelikož byly provedeny poměrně velké úpravy v celém programu, rozhodl jsem se v této kapitole popsat obecný postup vedoucí ke správné reprezentaci objektů. U některých skupin tvarů jsou sice tyto kroky doplněny například o další potřebné výpočty, ale tyto odlišnosti budou popsány až v konkrétních kapitolách.

První výraznou změnu přineslo vytváření nových objektů. V původní verzi byla tato problematika řešena tak, že pro získání nových bodů se nejprve zavolala metoda ***getInstance***, které byl předán seznam bodů s konkrétními souřadnicemi pro daný objekt. Pomocí těchto bodů se v metodě ***getInstance*** dopočítaly zbývající řídící body a výsledný seznam byl nakonec předán konstruktoru třídy. Díky tomu však, že jsem implementoval náhled vytvářeného objektu, není nezbytné už u vytváření nového objektu určovat přesné souřadnice řídících bodů. Metodu ***getInstance*** jsem tudíž využil jen pro přiřazení potřebného počtu řídících bodů k tvaru. O správné umístění bodů se stará nově přidaná metoda ***resize***, která při každém překreslení v procesu vytváření objektu přepočítá umístění všech řídících bodů.

Přepsána byla také metoda ***clone*** pro vytváření kopií objektů. Využívána je při vytváření záloh jednotlivých tvarů pro účely animování. Jelikož se ale při animování vyžadují pouze informace o pozicích řídících bodů a použitých barvách, je nejprve zkopírován seznam bodů objektu a následně je zavolán konstruktor, kterému je kopie předána. Nová instance tudíž není úplnou kopií původního objektu, poněvadž neobsahuje stejné nastavení všech atributů. Důležité je však nastavení identifikačního čísla tvaru, podle kterého se provádí porovnávání kopií.

Hlavní částí třídy pro každý tvar je pak samotný postup při vykreslování objektu. Pro tyto účely je vytvořena metoda ***paintShape***, které je předána instance třídy **Graphics2D**. Pomocí ní je umožňováno vykreslování do komponent, v našem případě je použita pro kreslení do editačního plátna. V prvním kroku se nejprve vytvoří nová instance konkrétního objektu, kterému jsou v konstruktoru předány potřebné parametry. V závislosti na typu vytvářeného objektu tvoří tyto hodnoty buďto souřadnice řídících bodů nebo například atributy vyjadřující výšku a šířku objektu. Dále se do zálohy uloží aktuální stav vykreslovacího objektu. Nejprve se vykresluje stín objektu, ovšem jen v případě, je-li tato možnost zvolena. Následující část kódu se zabývá barvou výplně. Jestliže není povolena přechodová barva, přiřadí se vykreslovacímu objektu jednoduchá barva zvolená uživatelem, v opačném případě je instanci **Graphics2D** přiřazen objekt třídy **GradientPaint**, jež nastavuje barevný přechod, tento přechod začíná od levého kraje objektu a končí u kraje pravého. Dále se už vykresluje konkrétní tvar, popřípadě jeho výplň. Následuje vykreslení ohraničení objektu. Všechny tyto kroky musí proběhnout ve správném pořadí, aby bylo zajištěno korektní překrývání těchto vrstev. Aby bylo dosaženo vrácení nastavení vykreslování do původního stavu, je použita předem vytvořená záloha. Celý tento proces je znázorněn níže na konkrétním diagramu.




*Obrázek 5 .1: Diagram popisující průběh metody pro vykreslování*

## 5.3 Křivky a úsečky

U těchto objektů byla složitost implementace nejjednodušší ze všech zde jmenovaných skupin. Jedná se totiž o objekty, jejichž tvar je definovaný pouze množinou řídících bodů, tudíž zde nemusí být řešeno počítání s natočením objektů, nebo jejich zkosením.

### Line (úsečka)

Tato třída reprezentuje tvar úsečky a je tvořena pomocí instance třídy **Line2D.Double**. Jejími hlavními atributy jsou objekty třídy **ShapePoint** udávající pozice koncových bodů.

### QuadCurve (kvadratická Bézierova křivka)

Tato třída slouží k reprezentování kvadratické Bézierovy křivky. Pro její vykreslení byla použita předdefinovaná třída **QuadCurve2D.Double**. Jako atributy jsou zde uvedeny tři instance třídy **ShapePoint**, které reprezentují řídící body této křivky. V důsledku použité metody vytváření objektů, kdy je možné přesně určit pouze dva body, jsem se rozhodl, že při vkládání nové kvadratické křivky budou definovány hlavně pozice koncových bodů. Všechny body však lze ještě dodatečné přemisťovat pomocí parametrů v editačním panelu.

### CubicCurve (kubická Bézierova křivka)

Tato třída slouží k vytváření kubických Bézierových křivek. Pro tento účel byla použita třída **CubicCurve2D.Double**. Celková funkcionalita je velmi podobná výše zmíněné kvadratické Bézierové křivce až na fakt, že kubická křivka je definována dvěma koncovými body a dvěma body řídícími. Podobně jako u křivky kvadratické, jsou i zde při vytváření nového objektu přesně definovány pouze souřadnice koncových bodů. Pozice všech bodů je však možné dodatečně měnit pomocí editačního panelu.

## 5.4 Předdefinované geometrické objekty

Implementace těchto tvarů představovala velmi složitou operací. Definování těchto objektů je totiž realizováno pomocí levého horního bodu a celkové výšky a šířky tvaru. Při vytváření tudíž nejde jednoduše nastavit natočení tohoto objektu, a tak musely být přidány k obecnému algoritmu dodatečné výpočty, které byly umístěny na začátek vykreslovací metody. Nejdříve byla vypočtena výška a šířka objektu, která se získala jako vzdálenost určených řídících bodů. Poté byl přepočten úhel, ve kterém se objekt nachází. V následném kódu je vyřešena situace, kdy se provádí rotace objektu kolem označeného řídícího bodu. Rotace objektu je totiž realizována pomocí natočení vykreslovací komponenty typu **Graphics2D**. Jelikož jsou však tyto geometrické tvary definovány pouze horním levým bodem, ale střed otáčení může být jakýkoliv řídící bod, je nejprve nutné souřadnice levého horního bodu přepočítat tak, aby odpovídaly požadované rotaci. Přepočet se tudíž provádí v závislosti na právě označeném bodu. Po této operaci již může být použito vykreslování pomocí natočení vykreslovací komponenty a vytvoření příslušné instance objektu, které jsou předány upravené souřadnice levého horního bodu a výška a šířka tvaru.

Za vysvětlení ještě určitě stojí proces zkosení, implementovaný u této skupiny tvarů. Jedna z odlišností je výpočet výšky objektu. Jelikož totiž jde o zkosený objekt, není už jeho výška definována jako vzdálenost určitých řídících bodů, ale nutno tuto hodnotu příslušně upravit pomocí

pythagorovy věty. Schéma pro tento výpočet je uvedeno níže. Druhou změnou je část kódu, která již přímo realizuje transformaci zešikmením. Proces je prakticky podobný průběhu metody ***shearShape*** implementované ve třídě **AbstractShape**. Za zmínění však stojí, že díky tomu, že zkosení je realizováno v již otočené souřadné soustavě, je zapotřebí pro posun použít upravených souřadnic levého horního bodu.

### Circle (kruh)

Tato třída slouží k vytvoření objektů reprezentující kruh. Tento tvar je sice svými vlastnostmi pouze speciálním typem elipsy, při jejím vytváření je použita právě třída **Ellipse2D.Double**, ale pro jeho časté využití jsem jej oddělil do samostatné třídy. Jejím základem je pět řídících bodů. Z čehož čtyři body leží přímo na ohraničení tohoto tvaru a pátý bod reprezentuje levý horní roh, který je zapotřebí při výpočtech. Jedinou podstatnou odlišností v algoritmu pro tento tvar je to, že místo parametrů pro výšku a šířku objektu je zde jen jedna hodnota reprezentující průměr kruhu. Jelikož je tento parametr počítán na základě šířky objektu, nemusí být při zkosení prováděna dodatečný přepočet, který upravoval hodnotu výšky.

### Ellipse (elipsa)

Tato třída reprezentuje tvary typu elipsa. Jde o objekt třídy **Ellipse2D.Double**, jehož hlavními parametry jsou výška a šířka. Podobně jako u implementace kruhu, i zde je pro manipulaci s tímto tvarem umístěno pět bodů. Čtyři leží na ohraničení elipsy a pátý bod tvoří levý horní roh objektu. Ten hraje klíčovou roli při výsledném vykreslení objektu.

### Rectangle (obdelník)

Tato třída slouží k vytváření tvarů typu obdélník a při její reprezentaci je použita právě třída **Rectangle2D.Double**. Hlavními parametry jsou zde výška a šířka objektu. A jako pozice pro čtyři řídící body jsou použity souřadnice vrcholů.

### RoundRectangle (zaoblený obdélník)

Tato třída tvoří objekty typu obdélník se zaoblenými rohy a při jeho vykreslování je použita třída **RoundRectangle2D.Double**. Implementace tohoto tvaru je velmi podobná implementaci obdélníku. I zde jsou totiž jako hlavní parametry použity výška a šířka objektu. Navíc je zde však přidán parametr udávající míru zaoblení rohů tohoto obdélníka. Řídící body potom leží na pozicích, na kterých by se nacházely vrcholy obdélníka, který by nebyl zaoblený.

### Oval (ovál)

Tato třída reprezentuje tvary typu ovál. Její implementace je prakticky shodná s implementací obdélníku se zaoblenými rohy. I zde je pro vykreslení tvaru použita třída **RoundRectangle2D.Double**, takže kromě výšky a šířky objektu je zde ještě požadována hodnota pro zaoblení. Ta je ovšem vypočítávána automaticky na největší přípustnou hodnotu, aby bylo dosaženo tvaru oválu.

### WrittenText (text)

Tato třída reprezentuje textové pole. Jejím základem je třída **Rectangle2D.Double**, která je také základem pro tvorbu obdélníku. I zde jsou tudíž řídící body umístěny do vrcholů tohoto nadefinovaného textového pole. Pro další editaci jsou také umožněny změny výšky a šířky tohoto pole. Hlavním účelem tohoto objektu je ale zobrazování textu, a tak obsahuje zmiňovaná třída ještě metody pro jeho editaci a následné korektní vykreslení. Kromě jednoduchých nastavení umožňující změnu a velikost fontu je zde přidána metoda pro parsování textu. Je totiž nutné, aby text nepřesáhnul hranice textového pole. Tato zmíněná metoda proto zajišťuje, aby se každý řádek, který by přesáhl hranice pole, rozdělil na více menších řádků. Popis hlavní myšlenky tohoto mechanizmu můžete najít na [4]. Další užitečnou vlastností je určitě také možnost zarovnávat text jak vertikálně, tak horizontálně.

## 5.5 Tvary tvořené pomocí Path2D s neměnnou strukturou

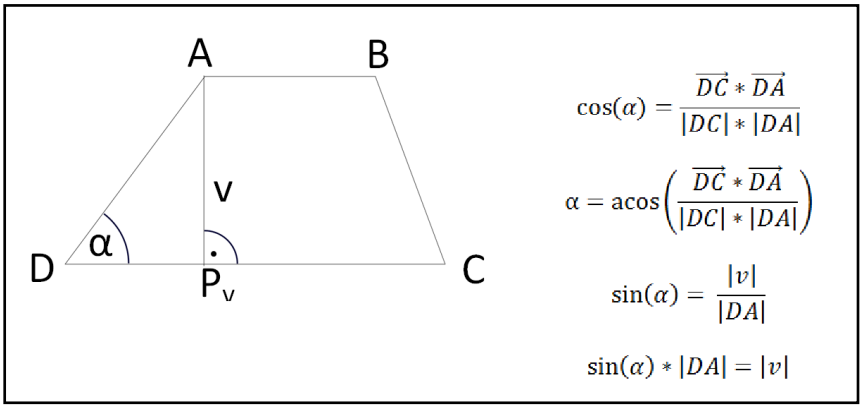
Implementace všech těchto tvarů se vyznačuje hlavně tím, že jsou vytvořeny za pomocí třídy **Path2D.Double** a řídící body jsou umístěny ve všech vrcholech požadovaného tvaru. Samotná vykreslovací metoda se díky těmto vlastnostem velice zjednoduší. Třída **Path2D.Double** totiž obsahuje metody díky kterým, můžeme propojovat libovolné dva body a to jak úsečkou, tak i křivkou. Takže pomocí metod této třídy můžeme jednoduše spojit všechny řídící body a nemusíme provádět žádné dodatečné výpočty, které by obstarávaly rotaci či zkosení objektu. Tyto transformace již totiž realizují samy řídící body objektu.

### Arrow (šipka)

Tato třída reprezentuje tvary typu šipka. Instance této třídy obsahuje sedm řídících bodů, které, jak již bylo zmíněno výše, tvoří všechny vrcholy tohoto tvaru. Algoritmus metody pro vykreslování je prakticky shodný s obecným algoritmem popsaným výše. Jedinou změnou je přítomnost části kódu, která se stará o vytvoření cesty přes všechny řídící body a je realizována právě pomocí třídy **Path2D.Double**. Doplňujícími parametry tohoto objektu jsou hodnoty určující velikost hrotu a těla šipky. Jejich nastavení se však neřeší v metodě pro vykreslování, ale už v metodě ***resize***, která určuje velikost a rozložení objektu.

### Trapezoid (lichoběžník)

Tato třída slouží k vytváření tvarů typu lichoběžník. Jak již bylo řečeno dříve, je tento tvar realizován pomocí třídy **Path2D.Double** a jeho řídící body jsou umístěny ve vrcholech tohoto lichoběžníku. Díky této vlastnosti je k obecnému vykreslovacímu algoritmu přidána jen část kódu, která obstarává vytvoření cesty přes všechny řídící vrcholy. Za zmínku však ještě stojí fakt, že se při této implementaci značně zkomplikoval výpočet výšky, která už nemohla být definována jako vzdálenost dvou bodů, ale pro její vyjádření bylo potřeba použít známá goniometrická pravidla. Schéma tohoto výpočtu je znázorněno níže.




*Obrázek 5 .2: Schéma výpočtu výšky pro lichoběžník*

## 5.6 Tvary tvořené pomocí Path2D s proměnlivou strukturou

Stejně jako předešlá skupina, i zde jsou všechny tvary vykreslovány pomocí třídy **Path2D.Double**, která umožňuje spojovat úsečkami, či křivkami nadefinované body. Řídící body však v tomto případě nejsou umístěny ve vrcholech tohoto tvaru, ale jsou tvořeny vždy jen čtyřmi body, které označují pole, ve kterém se objekt nachází. Důvodem tohoto postupu je fakt, že počet řídících bodů by měl zůstat neměnný. Počet vrcholů objektu se však mění v závislosti na uživatelem zadaném parametru. Díky této vlastnosti je vykreslovací metoda rozšířena o velké množství kódu zajišťující korektní nadefinování pozic jednotlivých vrcholů objektu. Na rozdíl od předcházející skupiny objektů, zde musely být samozřejmě zohledněny modifikace natočení nebo zkosení objektu.

### Ngon (n-úhelník)

Tato třída reprezentuje všechny tvary typu n-úhelník. Jak již bylo řečeno v úvodu, řídící body definují pouze ohraničující pole pro tento tvar. Do těla vykreslovací metody bylo tudíž nutné implementovat obecný algoritmus, který se staral o správné vytvoření celkového tvaru v závislosti na zvoleném počtu vrcholů. V první fázi se nejprve vypočítají pozice vrcholů pravidelného n-úhelníku a následně jsou tyto souřadnice ještě dodatečně upraveny v závislosti na poměru stran vytvořeného ohraničujícího pole.

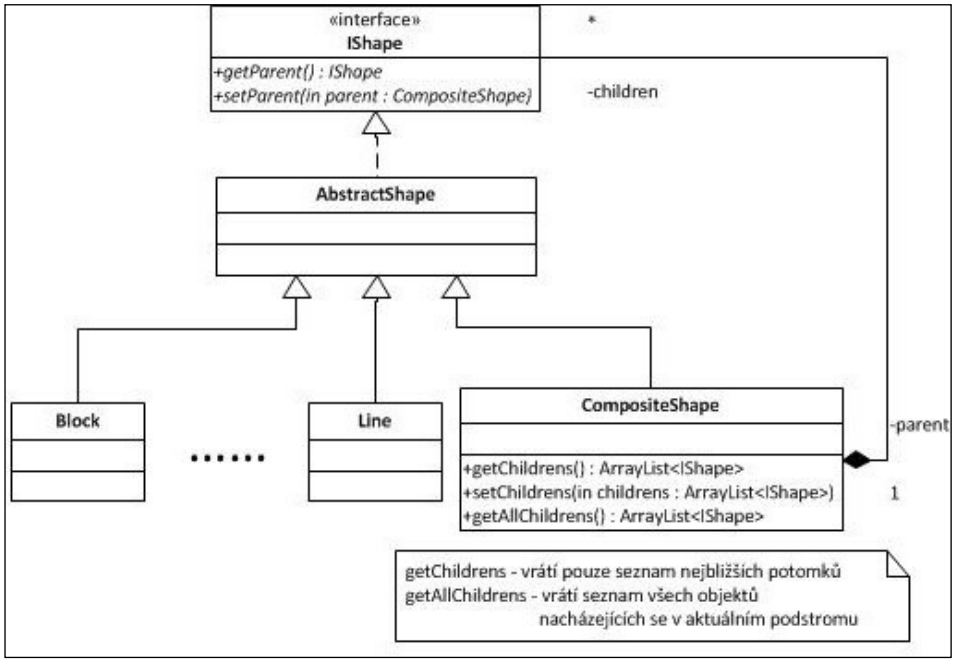
### Star (hvězda)

Tato třída slouží k vytváření tvarů typu hvězda. Díky faktu, že řídící body určují pouze ohraničující pole tohoto objektu, je v těle vykreslovací metody umístěn algoritmus pro vypočítání správných souřadnic vrcholů v závislosti na zvoleném počtu cípů hvězdy a na použité hodnotě pro velikost její středové kružnice. V první fázi se nejprve vypočítají souřadnice pro pravidelnou hvězdu a následně jsou souřadnice těchto bodů ještě upraveny tak, aby odpovídaly poměru stran vytvořeného ohraničujícího pole.

## 5.7 Složený objekt

### CompositeShape (složený objekt)

Tato třída reprezentuje veškeré sloučené objekty. Na rozdíl od uvedených objektů, vykreslovací metoda neobsahuje ani výše popsaný obecný algoritmus, ale je zde umístěna pouze metoda na přepočítání úhlu rotace. Rozhodl jsem se totiž, že složený objekt bude fungovat jen jako záštita pro jakoukoliv manipulaci s objekty, ale samotný objekt nebude nijak vykreslován. V původní verzi tato třída fungovala tak, že při shlukování se propojila celá geometrie těles. Problém by však v tomto případě byl při potřebě získání původních základních objektů. Rozhodl jsem se proto k tomu, aby při shlukování bylo pouze vytvořeno ohraničující pole kolem daných objektů, které bylo pro tyto shlukované objekty jako řídící. Aby bylo tohoto cíle dosáhnuto, musely být upraveny metody jak pro vybírání objektů, tak i pro jejich transformace. Každému tvaru jsem proto přiřadil atribut nesoucí odkaz na rodičovský objekt, neboli na objekt, který jej zaštiťuje. Pomocí tohoto parametru se pak kontroluje, zda nemá označovaný objekt přiřazený sloučený objekt, který by měl v takovém případě v označení přednost. Co se týče transformací, obsahuje složený objekt ještě atribut, vyjadřující seznam odkazů na všechny obsažené tvary. Pomocí tohoto parametru je tak možné modifikovat nejen daný složený objekt, ale také všechny jeho potomky. Jak je asi zřejmé, při implementaci tohoto tvaru jsem využil návrhový vzor Kompozit, jehož popis je detailně uveden v [6].

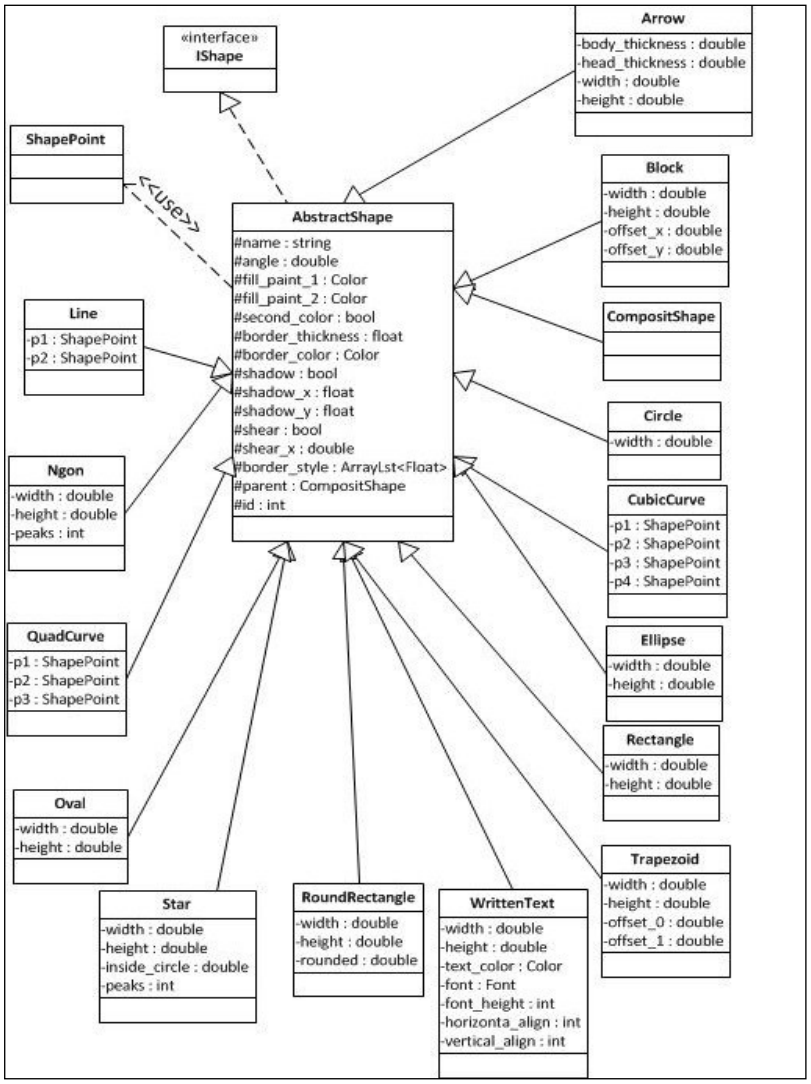



*Obrázek 5 .3: Třídní diagram zachycující použití návrhového vzoru Kompozit u objektů typu CompositeShape*

## 5.8 Tvary reprezentující prostorová tělesa

### Block (kvádr)

Tato třída tvoří tvary typu kvádr a jako jediná ze zde přidaných tvarů reprezentuje prostorové těleso. Základem tohoto objektu je přední stěna, v jejíchž vrcholech jsou umístěny čtyři řídící body. Pro projekci tohoto prostorového objektu do roviny bylo použito kosoúhlé promítání, poněvadž jsou zde zachovány vlastnosti, jako je rovnoběžnost a poměr stran. Celkový vzhled tělesa pak udávají parametry určující odsazení zadní stěny objektu. Jelikož se jedná o prostorové těleso zobrazované do roviny, obsahuje vykreslovací metoda ještě velké množství dodatečných výpočtů. Není třeba totiž jen vypočítat všechny vrcholy tohoto tělesa, ale je třeba zohlednit i viditelnost všech stěn a to nejen v závislosti na nastavení odsazení zadní stěny, ale také na aktuálním úhlu natočení tohoto objektu. Pro lepší manipulaci jsem se tak rozhodl celé těleso reprezentovat pěti samostatnými objekty typu **Path2D.Double**, které představují všechny možné viditelné stěny.




*Obrázek 5 .4: Třídní diagram zohledňující přístupné atributy objektů*

# 6 Modifikační panely

Pro práci s vlastnostmi vložených objektů jsem se rozhodl vytvořit levý postranní panel, který by se přizpůsoboval označenému objektu. Poskytoval by tak uživateli ihned výčet parametrů, které by mohl měnit a nemusel by je tak složitě hledat. Při zvolení tohoto konceptu jsem se nechal inspirovat aplikací Blender, která používá právě pro modifikaci objektů postranní panely.

Pro implementaci jsem použil postup, kdy modifikační panel sám o sobě zůstává pořád jako jedna a ta samá instance, ale jeho obsah určují níže popsané třídy. Při označení objektu se nejprve vymaže celý obsah panelu, poté se vybere příslušná třída a zavolá se její metoda ***reBuild***, která se postará o naplnění panelu konkrétními komponentami.

## 6.1 Rozhraní a řídící třída pro modifikační dialogy

### IModifyDialog

Toto rozhraní je společné pro všechny konkrétní modifikační dialogy. Definuje se zde metoda ***reBuild***, která se stará o naplnění modifikačního panelu komponentami. Metodě je také poslán odkaz na označený objekt, jehož parametry jsou v komponentách zobrazeny. Kvůli nutnosti dialogu reagovat na události stisknutí tlačítka, dědí **IModifyDialog** od rozhraní **ActionListener**.

### DialogFactory

**DialogFactory** je řídící třídou pro konkrétní modifikační dialogy objektů. Pomocí této třídy se právě rozhoduje, který dialog má být do panelu vykreslen. Hlavním atributem této třídy je objekt typu *HashMap*, který slouží jako kontejner pro všechny dostupné druhy modifikačních dialogů. Ten je naplněn při zavolání konstruktoru. Jako klíč je zde vždy uveden řetězec, kterým je definován typ tvaru, a jako hodnota je k němu přiřazena instance příslušného modifikačního dialogu. Při označení objektu se zavolá metoda ***getDialog***, která prohledá kontejner s nahranými druhy dialogů. U příslušného záznamu se nakonec zavolá metoda ***reBuild***, která se stará o korektní vykreslení panelu. Pokud však není označen žádný objekt nebo je naopak označeno více objektů, o vykreslení je požádána instance třídy **NothingDialog**. Kvůli relativně velké paměťové náročnosti je zde použit návrhový vzor singleton. Tím se i zrychlí celá aplikace, poněvadž nebude třeba opětovné inicializace celého kontejneru s dialogy.

## 6.2 Jednotlivé modifikační dialogy objektů a jejich specifikace

V této části bych chtěl popsat třídy reprezentující modifikační dialogy pro jednotlivé typy tvarů. Pro lepší přehlednost zde ale nejdříve nadefinuji základní nastavení, která jsou společná pro valnou většinu objektů. Některé dialogy tyto nastavení rozšiřují o další prvky a některé je naopak neposkytují, tyto rozdíly však uvedu již u konkrétní třídy.

Základní nastavení:

- změna jména objektu
- nadefinování výšky a šířky objektu
- nadefinování natočení objektu
- zvolení možnosti pro vykreslení přechodové barvy výplně

- nadefinování barvy výplně a barvy přechodu (barva přechodu je použita jen v případě, že je označena možnost pro vykreslení přechodové barvy)
- nadefinování barvy, tloušťky a stylu ohraničení
- povolení možnosti vykreslovat stín objektu společně s nastavením jeho odsazení po ose x a ose y. Je třeba ještě podotknout, že průhlednost stínu se automaticky nastavuje v závislosti na průhlednosti použité barvy výplně
- povolení modifikace objektu zkosením společně s nastavením koeficientu zkosení
- posunutí objektu do popředí nebo do pozadí v seznamu vykreslovaných objektů (tento seznam poté určuje pořadí vykreslovaných objektů)

### ArrowDialog

Tato třída tvoří dialog pro tvar typu šipka. Kromě výše uvedených základních nastavení poskytuje uživateli dva další parametry. Jedním z nich je možnost nastavit velikost hrotu této šipky. Tento parametr udává, kolik procent šířky celkového tvaru má zabírat právě hrot šipky. Dalším parametrem lze nastavit velikost těla šipky, což udává, kolik procent výšky tvaru má zabírat tělo šipky.

### BlockDialog

Tato třída tvoří dialog pro tvar typu kvádr. Jako jediný přidaný objekt reprezentuje projekci prostorového objektu, což představuje větší náročnost na výpočet a celkovou stavbu objektu. Z tohoto důvodu objekt ze základních nastavení neobsahuje možnost zvolení přechodové barvy a modifikaci zkosením. Naopak ale umožňuje nastavení odsazení na ose x a ose y, kterými lze ovlivnit prostorový vzhled celého objektu.

### CircleDialog

Tato třída tvoří dialog pro tvar typu kruh. Obsahuje všechny výše uvedené základní nastavení až na výjimku nastavení výšky a šířky objektu. Tyto dva parametry jsou nahrazeny údajem typu průměr. Kruh sice lze definovat i pomocí elipsy, ale kvůli jeho častému používání jsem jej raději reprezentoval jako samostatný tvar.

### CompositeDialog

Tato třída reprezentuje dialog pro složené objekty. Díky tomu, že jde o seskupení různých objektů, bylo by složité a možná i zbytečné u nich definovat jakákoliv nastavení. Proto poskytuje dialog pro složený objekt pouze možnost změny jména a nastavení natočení tohoto seskupení objektů.

### CubicCurveDialog

Tato třída tvoří dialogové okno pro kubickou křivku. Aplikují se zde všechna základní nastavení až na jisté výjimky. Jelikož jde o křivku, chybí zde nastavení výšky a šířky objektu. Místo nich je zde obsaženo nastavení umístění všech čtyř řídících bodů. Dále zde chybí nastavení pro zkosení objektu, to by totiž u tohoto objektu bylo zbytečné.

### EllipseDialog

Tato třída reprezentuje dialogové okno pro objekty typu elipsa. Jelikož jde o objekt, u kterého nejsou vyžadovány žádné speciální parametry, obsahuje toto dialogové okno pouze základní nastavení.

### LineDialog

Tato třída tvoří dialogové okno pro objekty typu úsečka. Jelikož by u tohoto objektu bylo nepraktické určovat jeho výšku a šířku, je toto nastavení nahrazeno definováním souřadnic koncových bodů. Dále zde chybí nastavení pro zešikmení. Kromě těchto omezení jsou zde aplikovány všechny základní nastavení.

### NgonDialog

Tato třída slouží k vytvoření dialogu pro tvar typu n-úhelník. Obsaženy jsou zde všechny základní nastavení. Kromě nich je zde však přidán navíc parametr pro určení počtu vrcholů.

### NothingDialog

Tato třída slouží k vytvoření takzvaného nespecifikovaného dialogu. Není v něm obsaženo žádné nastavení, poněvadž je využíván při situaci, kdy není označen žádný objekt, nebo naopak je označeno více objektů najednou.

### OvalDialog

Tato třída umožňuje vytvoření dialogu pro tvar typu ovál. Jelikož u tohoto tvaru nejsou požadovány žádné specifické parametry. Obsahuje dialogové okno pouze základní nastavení.

### QuadCurveDialog

Tato třída reprezentuje dialogové okno pro tvary typu kvadratická křivka. Stejně jako například u již zmíněné úsečky nebo kubické křivky, i zde by bylo nastavení výšky a šířky objektu nepraktické. Proto je toto nastavení nahrazeno specifikací souřadnic jednotlivých řídících bodů. Dále je zde vypuštěno nastavení zkosení objektu, protože by pro tento typ objektu nemělo žádný větší význam.

### RectangleDialog

Tato třída tvoří dialog pro nastavení tvaru typu obdélník. Jelikož jde o objekt, který nevyžaduje žádné speciální parametry, je u tohoto dialogového okna obsaženo pouze základní nastavení.

### RoundRectangleDialog

Tato třída reprezentuje dialogové okno pro tvary typu obdélník s kulatými rohy. Jelikož tento tvar do jisté míry vychází z obdélníku, jsou i jejich dialogová okna velmi podobná. Obdélník s kulatými rohy tudíž obsahuje všechna základní nastavení. Kromě nich je zde však navíc použit ještě parametr udávající velikost zaoblení rohů.

### StarDialog

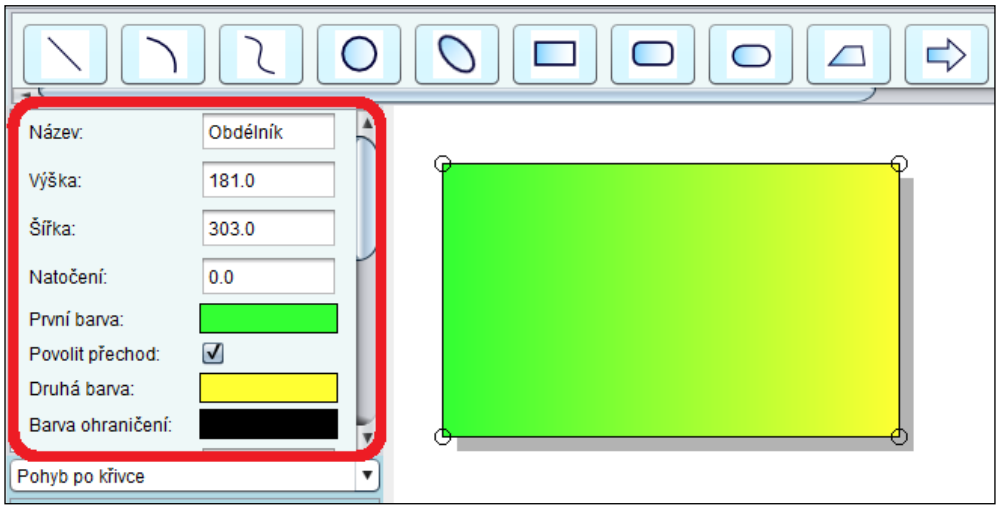
Tato třída představuje dialog pro tvary typu hvězda. Pro její modifikace může uživatel využívat všechna výše uvedená základní nastavení. Kromě nich však obsahuje tento tvar ještě dva další parametry. Jedním z těchto parametrů můžeme nastavit počet cípů dané hvězdy a druhým je možné ovlivňovat velikost středu označené hvězdy. Jelikož by se tento údaj dal jen stěží vyjádřit pro hvězdu, u které není zachován základní poměr stran, udává tato hodnota velikost procentuální velikost středu vzhledem k výšce a šířce objektu.

### TrapezoidDialog

Tato třída reprezentuje dialogové okno pro tvary typu lichoběžník. U tohoto objektu může uživatel využívat všechna základní nastavení. Kromě nich jsou zde však přítomny další dva parametry udávající odsazení bodů. Jak je totiž známo, lichoběžník obsahuje dvě základny. Použil jsem proto následující postup. Jedna základna je implementována jako fixní a její velikost je definována šířkou objektu. Při nulovém nastavení hodnot odsazení bodů bychom potom dospěli k obdélníku. Díky přidaným parametrům však můžeme libovolně posouvat vrcholy, jež nejsou řídícími body pro fixní základnu, a dosáhnout tak libovolného lichoběžníku.

### WrittenTextDialog

Tato třída tvoří dialogové okno pro textová pole. Tento tvar podporuje všechny základní nastavení a k nim přidává ještě další parametry. Jak by se již dalo očekávat, jedním z parametrů je text, který se bude v tomto poli zobrazovat. Dále jsou zde obsažena ještě nastavení fontu a to konkrétně určení jeho velikosti a stylu. Nabídka stylů obsahuje všechny dostupné systémové fonty. Navíc je uživateli nabídnuta ještě možnost zarovnání textu a to jak v horizontálním směru (vlevo, na střed a vpravo) tak ve směru vertikálním (nahoru, na střed, dolů).




*Obrázek 6 .1: Ukázka modifikačního dialogu objektu*

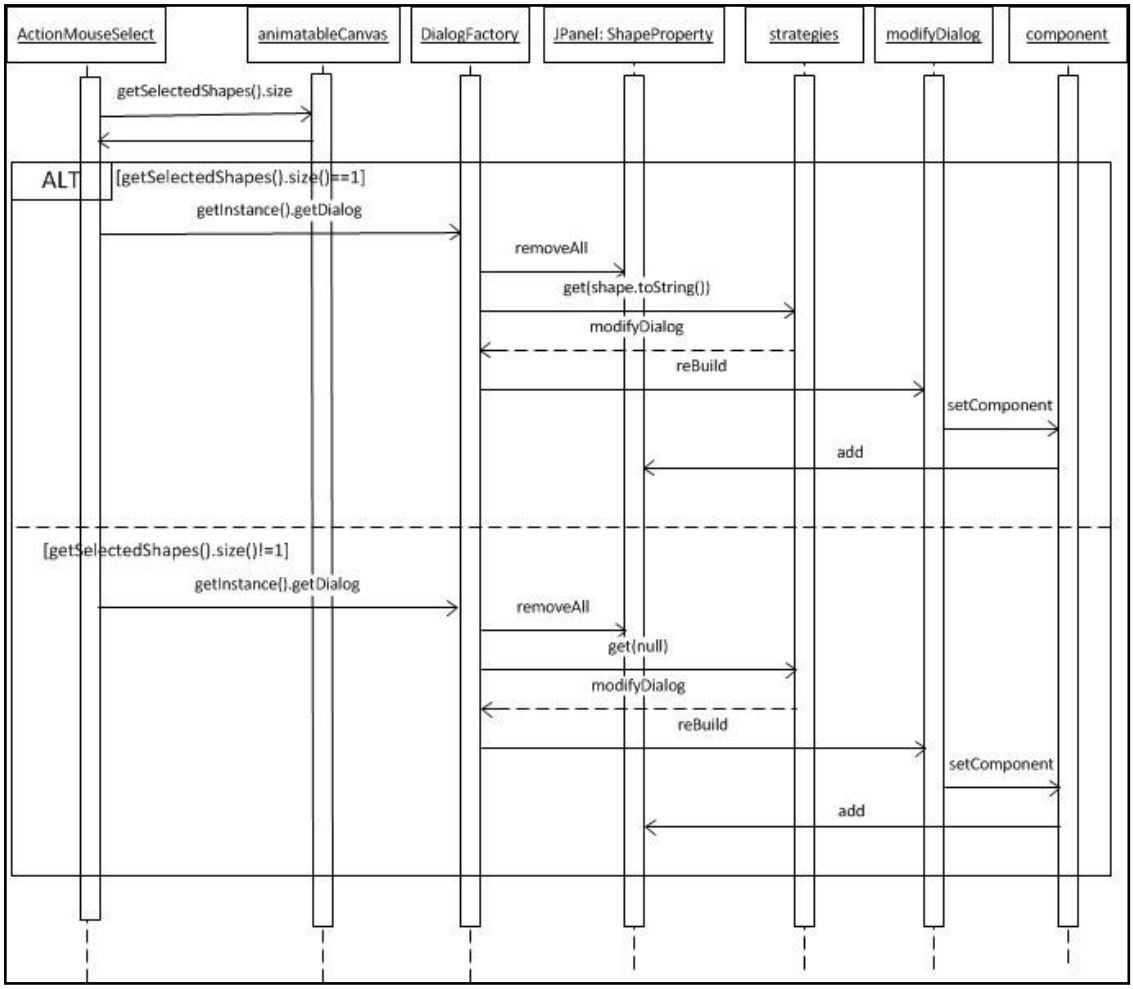
# 7 Komponenty pro modifikační panel objektu

Pro komponenty použité u modifikačních panelů objektů jsem se rozhodl vytvořit nový systém. Přidávání jednotlivých grafických prvků jako například *JLabel*, nebo *JTextField* se v pozdější fázi vytváření programu ukázalo jako velmi rozsáhlé a výsledný kód se stal potom velmi nepřehledným. Navíc by bylo dosti komplikované psát nové modifikační panely nebo modifikovat panely stávající o nové prvky. Vytvořil jsem proto samostatný balík se třídami, které tyto nedostatky řeší. Hlavní myšlenkou tohoto systému je použití reflexe. Díky ní potom stačí pro programátora, který bude tuto aplikaci rozšiřovat, aby vytvořil instanci jedné z následujících tříd a přiřadil k ní vlastnost tohoto objektu. Vlastnost samozřejmě musí obsahovat veřejné ***get*** a ***set*** metody, pomocí kterých se bude přistupovat k datům. Jednotlivé komponenty pracují s obecnými tvary typu **IShape**. Díky tomu jsou komponenty jednotné a použitelné pro všechny objekty, které rozhraní **IShape** implementují. Pomocí použité reflexe však nemusí řešit přetypování, ale mohou velmi jednoduše přistupovat i k vlastnostem, které jsou unikátní pro jednotlivé tvary a tudíž nejsou přes rozhraní **IShape** viditelné.

Každá z těchto uvedených tříd v podstatě obsahuje dvě hlavní metody.

Jednou z nich je konstruktor, kde se inicializují veškeré proměnné, které se v dané chvíli inicializovat mohou. Inicializace patří totiž k časově nejnáročnějším operacím, a tak jsem právě co nejvíce proměnných přesunul do konstruktoru, který se pro danou instanci volá jen jednou. Tím jsem docílil rychlejšího chodu celé aplikace.

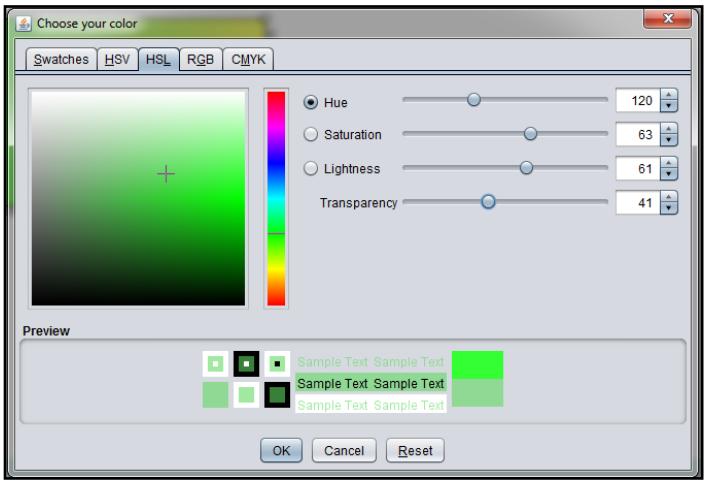
Druhá hlavní metoda se nazývá ***setComponent***, ta se volá při každé aktualizaci panelu. Prvním účelem této metody je zajištění správného rozložení základních grafických komponent. To je zajištěno pomocí odkazu na modifikační dialog a odkazu na objekt typu *GridBagConstraints*, který se stará o pozicování v celém modifikačním dialogu. Druhým účelem této metody je naplnění těchto komponent korektními hodnotami. Zde se právě využívá již zmiňovaná třída **PropertyUtils** a s ní spojená reflexe. Tato třída totiž poskytuje statickou metodu ***getProperty***, která dokáže pomocí odkazu na objekt a názvu jeho vlastnosti vrátit hodnotu této vlastnosti, a to i přes to, že není například kvůli použitému rozhraní viditelná. Na druhou stranu poskytuje také statickou metodu ***setProperty***, která dokáže nastavit hodnotu vlastnosti pomocí odkazu na určitý objekt, názvu jeho vlastnosti a požadované hodnoty. Samozřejmě je vyžadováno, aby objekt tuto vlastnost měl naimplementovanou, v opačném případě se vyvolá výjimka.




*Obrázek 7 .1: Sekvenční diagram zobrazující obecný postup volání tříd a objektů pro vykreslení požadovaného informačního dialogu při akci označení objektu.*

## 7.1 ColorComponent

Tato komponenta se stará o veškerá nastavení spojená s barvou. Je složena z dvou grafických komponent typu *JLabel*. První komponenta představuje doprovodný text a druhá obsahuje náhled se zvolenou barvou. Na náhledu je aplikován posluchač, který po stisknutí vyvolá komponentu *JColorChooser*. Tato komponenta dovoluje uživateli vybrat požadovanou barvu. A to buďto pomocí vzorníku, nebo barevných modelů HSV, HSL, RGB nebo CMYK a to včetně nastavení jejich průhlednosti. Po zvolení barvy je změna ihned aplikována na vybraný tvar.




*Obrázek 7 .2: Použitá grafická komponenta JColorChooser*

## 7.2 CheckComponent_noAcept

Tato komponenta se stará o nastavení, která jsou reprezentována pomocí datového typu *boolean*. Obsahují dvě grafické komponenty. Jednou z nich je *JLabel*, který reprezentuje doprovodný text a druhou z nich je *JCheckBox*, který reprezentuje samotnou hodnotu vlastnosti. Důležitým aspektem je, že změny této hodnoty nejsou vykonány ihned, ale je třeba je aplikovat pomocí potvrzovacího tlačítka Potvrdit. Využívána je především v případech, kdy změna ovlivňuje geometrii tvaru.

## 7.3 CheckComponent

Tato komponenta se stará, podobně jako již zmíněná komponenta **CheckComponent_noAcept**, o nastavení vlastností, reprezentovaných pomocí datového typu *boolean*. I zde je obsažena komponenta *JLabel* pro zobrazení doprovodného textu a komponenta *JCheckBox* pro reprezentaci hodnoty požadované vlastnosti. Rozdílem je zde však to, že při změně hodnoty se vyvolá událost, která tuto operaci ihned promítne do nastavení objektu. Využívána je především u vlastností, které nikterak neovlivňují geometrii tvaru.

## 7.4 NameComponent

Tato komponenta je základní komponentou a složí pro zobrazení a nastavení jména objektu. Obsahuje komponentu *JLabel* pro doprovodný text a komponentu *JTextField* pro samotné zobrazení jména či jeho přepsání. Jako jediná komponenta neobsahuje žádnou reflexi, poněvadž je specializovaná pouze na jméno objektu, které je dostupné i přes rozhraní *IShape*.

## 7.5 NumberComponent

Tato komponenta se stará o veškerá nastavení, která jsou reprezentována pomocí číselných hodnot. Obsahuje komponentu *JLabel* pro zobrazení doprovodného textu a komponentu *JTextField* pro reprezentaci číselné hodnoty požadované vlastnosti. Díky statické metodě getPropertyType, kterou poskytuje třída **PropertyUtils**, jde totiž rozpoznat jakého datového typu požadovaná hodnota je. Pomocí této vlastnosti lze třídu **NumberComponent** použít pro reprezentaci hodnot typu *integer*, *float*

nebo *double*. Pro lepší přehlednost se u hodnot s plovoucí desetinnou čárkou provádí zaokrouhlení na dvě desetinná místa.

## 7.6 PositionComponent

Tato komponenta se stará o nastavení pozice bodu a patří mezi nejsložitější použité komponenty. Obsahuje tři grafické komponenty typu *JLabel* a dvě komponenty typu *JTextField*. První objekt třídy *JLabel* reprezentuje popis použitého bodu, následující dva pak vyjadřují označení souřadnice. K těmto dvěma označením pak náleží komponenty *JTextField*, obsahující příslušné hodnoty souřadnic.

## 7.7 TextAreaComponent

Tato komponenta umožňuje zapisovat nebo zobrazovat text z komponenty. Obsahuje dvě základní grafické komponenty. Jednou z nich je doprovodný text typu *JLabel* a druhou je samotné textové pole typu *JTextArea*, které pomocí použití *JScrollPane* podporuje i vkládání textu o větších rozměrech. Jediné použití je zatím u tvaru typu **WrittenText**, ale s ohledem na další možné rozšiřování vlastností celé aplikace jsem ji izoloval jako samostatnou komponentu.

## 7.8 ComboBoxComponent

Tato komponenta umožňuje uživateli aplikace vybrat si nastavení z předem předpřipravených možností. Pro reprezentaci byly použity opět dvě grafické komponenty z knihovny grafických prvků Swing. Jako u ostatních komponent je zde použit objekt třídy *JLabel* pro reprezentaci doprovodného textu. Pro samotný výběr možností je zde použita komponenta typu *JComboBox*. Pro naplnění výběrového pole je vyžadována datová struktura typu *LinkedHashMap*, kde klíče jsou obecného datového typu *Object* a reprezentují jednotlivá nastavení. Hodnoty jsou pak typu *String* a reprezentují názvy jednotlivých nastavení.

# 8 Seznam objektů

## 8.1 Požadavky

Dalším bodem bakalářské práce byl požadavek na možnost zobrazení vložených objektů včetně zobrazování jejich stromové struktury. Do aplikace byla totiž přidána možnost shlukování objektů pomocí objektu **CompositeShape**. Proto je žádoucí, aby daný seznam nezobrazoval pouze výčet všech tvarů, ale aby respektoval i jejich hierarchickou strukturu. Tato vlastnost je důležitá především při akci shlukování objektů, kdy se vytvoří nový objekt třídy **CompositShape**. V seznamu tvarů se pak tato instance zobrazí jako nový adresář, který pod sebou obsahuje shlukované tvary. Tímto přístupem byla uživateli poskytnuta větší přehlednost nad přítomnými objekty.

## 8.2 Implementace

Panel se seznamem objektů byl umístěn na pravé straně aplikace pro jednoduchou přehlednost. Tento koncept je totiž obsažen i například u aplikace Blender, který taktéž používá seznam se zobrazováním hierarchické struktury obsažených objektů. Samotná implementace panelu se seznamem objektů je obsažena ve třídě **MainDialog** v metodě ***initGUI***.

Všechny komponenty související se seznamem objektů jsou sloučeny pod objektem seznam typu *JPanel*. Samotný seznam objektů je pak z důvodu jeho možné velké velikosti vložen do objektu třídy *JScrollPane*. Pro implementaci celého seznamu objektů byly použity následující čtyři třídy.

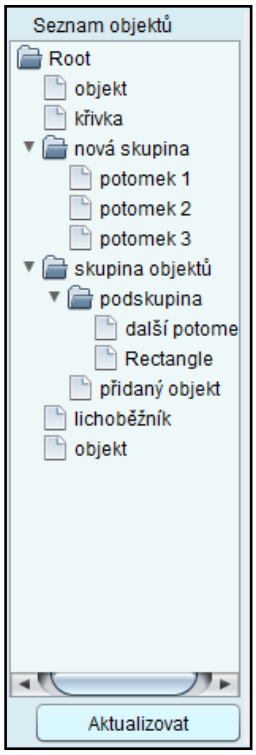
Třída *JTree* slouží k zobrazování hierarchických dat. Jako vstupní parametr konstruktoru byl v aplikaci použit objekt třídy *DefaultTreeModel*, pro lepší manipulaci s daty. Dále bylo nutno použít implementovanou metodu ***setCellRenderer***, pomocí které se nastavuje, jak se mají data uvedené struktury používat. Další popis vlastností těchto dvou jmenovaných tříd je popsán níže.

Třída *DefaultTreeModel* je datový model implementující rozhraní *TreeModel*. Díky této třídě dokáže grafická komponenta *JTree* přistupovat k datům a pomocí ní také ví, jak mají být jednotlivé objekty typu *TreeNode* zobrazovány.

Třída *DefaultMutableTreeNode* slouží k vytvoření univerzálního uzlu stromové datové struktury a jeho propojení s uzlem rodiče, či uzly potomků. Má-li uzel alespoň jednoho potomka, je potom zobrazen jako adresář. V opačném případě je zobrazen jako list. Každý vytvořený uzel také nese odkaz na objekt, ke kterému náleží, což může být použito pro vybírání objektu pomocí označení záznamu v seznamu objektů.

Třída **ShapeTreeCellRendererComponent** slouží k zobrazování jmen objektů. Klasická defaultní implementace totiž používá k zobrazování názvů v *JTree* metodu ***toString***. Ta je ovšem již použita pro zjištění typu tvaru. Třída **ShapeTreeCellRendererComponent** proto dědí z třídy *DefaultTreeCellRenderer* a přepisuje její metodu ***getTreeCellRendererComponent***, která se stará právě o zobrazované názvy uzlů ve stromu. Tato metoda je přepsána tak, že pro zjišťování názvu používá místo metody ***toString***, metodu ***getName***. Detailnější popis této problematiky i jeho řešení lze najít na [5].

Pro vytvoření stromového seznamu slouží metoda ***createTree*** vyvolávána tlačítkem Aktualizovat. Metoda nejprve vymaže celou předcházející strukturu a ze seznamu tvarů vybere všechny tvary, které nemají žádného rodiče. Tyto tvary tvoří první hladinu seznamu. Pro každý takto vybraný tvar se vytvoří příslušný uzel a zavolá se metoda ***check_childrens***. Je-li objekt, u kterého byla metoda volána, typu **CompositeShape**, vytvoří pro každého jeho potomka nový uzel a iteračně u nich opět zavolá metodu ***check_childrens***. Tímto je zaručeno vytvoření korektního stromového seznamu pro požadované účely. Ukázka takovéhoto seznamu v aplikaci je možno vidět na následujícím obrázku.




*Obrázek 8 .1: Ukázka implementovaného seznamu objektů*

# 9 Úpravy původní aplikace

## 9.1 Implementace náhledu tvořeného objektu

Jak již bylo předesláno v dřívější kapitole, pro dosažení lepší práce s programem bylo nutné nahradit původní model vytváření objektů. Byla zde totiž použita metoda, která pomocí bodu stisku a bodu uvolnění tlačítka, následně vytvořila objekt. Tento princip jsem musel nahradit sofistikovanějším přístupem, kdy jsem už při stisku tlačítka zavolal metodu pro vytvoření objektu. A při každém posunu ukazatele myši se zavolala tato metoda znovu. Potřebné dva body pro vytvoření objektu představovaly body stisku a aktuální pozice ukazatele. Tím bylo dosaženo to, že byl objekt zobrazován ještě před uvolnění tlačítka myši. Podrobný popis algoritmu naleznete v dalších odstavcích.

Samotnou třídu implementující tuto funkčnost můžete najít v balíčku **dialog.dlgMain.action.mouseAction** pod jménem **ActionMouseDraggCreate**. Hlavní část třídy představuje přepsaná metoda ***execute***, která se stará o samotné vykonávání akce. Průběh tohoto procesu by se mohl rozdělit na dvě fáze.

První fázi představuje první průchod metodou. Nejprve se získají potřebné informace o pozicích pomocných bodů. Objekt pressPoint představuje bod stisku tlačítka, objekt releasePoint představuje bod, ve kterém se zrovna ukazatel nachází. Následně se získá seznam všech aktuálně označených objektů a u každého označeného objektu se zavolá metoda ***resize***. V prvním průchodu však tento krok nemá význam, protože při přechodu aplikace do stavu tvoření objektu, odznačí se automaticky všechny objekty. Seznam je proto prázdný. V druhé části metody se už ovšem vytvoří nový objekt podle označeného tvaru a pozic pomocných bodů. Tento vytvořený tvar se na konci první iterace označí. To je klíčové pro další průchody touto metodou.

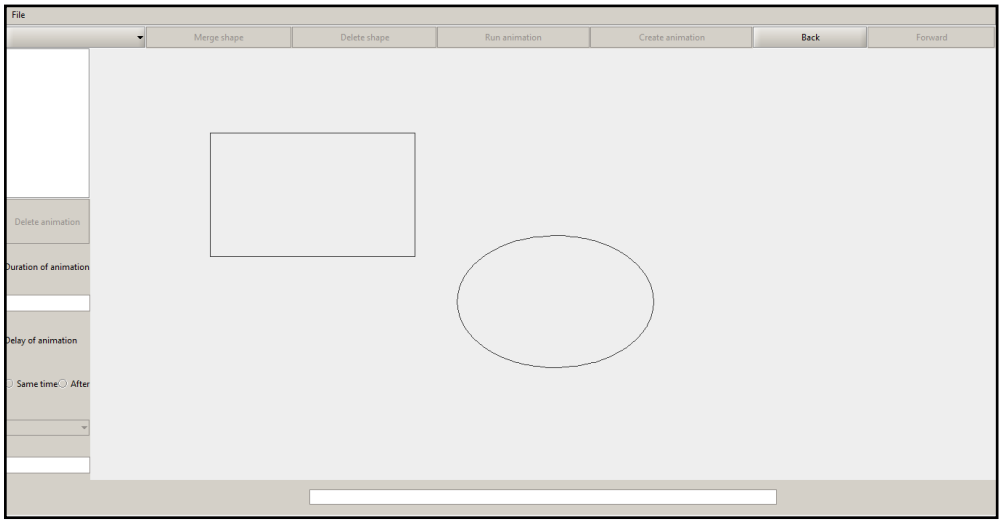
Druhá fáze představuje všechny následující průchody metodou až po uvolnění tlačítka. Na začátku se opět získají informace o pozicích bodů a získá se seznam označených tvarů. Ten však již obsahuje vytvářený objekt, který byl označen v prvním průchodu. Na tento objekt se tudíž již zavolá metoda ***resize***, která obstarává změnu velikosti objektu. Následující kód, který v prvním průchodu vytvářel novou instanci tvaru, je již nyní přeskočen, poněvadž seznam označených objektů již není prázdný. Tímto postupem bylo docíleno požadovaného náhledu objektu.

## 9.2 Vylepšení grafického uživatelského rozhraní

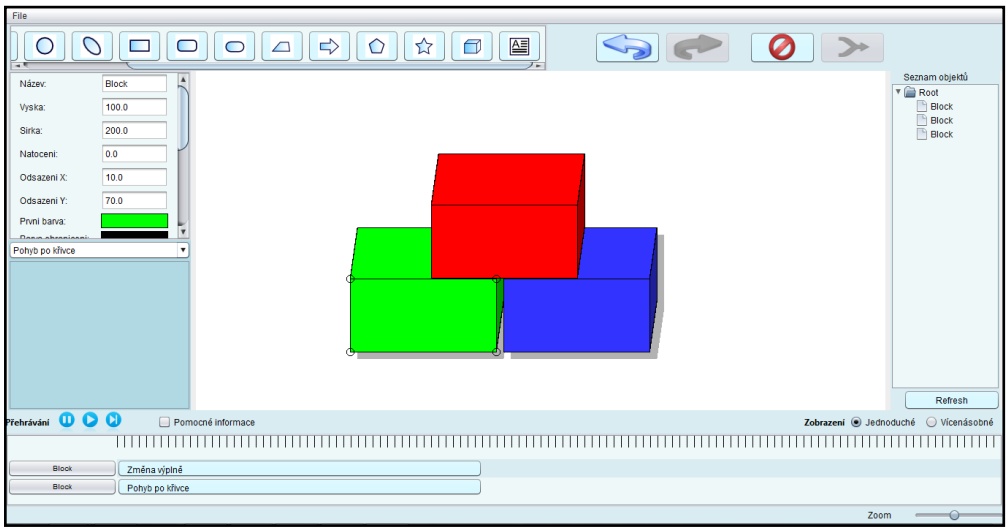
Jednou z dalších částí této bakalářské práce bylo vylepšení celkového grafického uživatelského rozhraní aplikace. Jak již bylo zmíněno výše, byl už do programu například přidán panel pro editaci vlastností a seznam objektů přítomných na kreslícím plátně. Kromě těchto přidaných komponent však bylo třeba vylepšit i celkový vzhled aplikace a to například vytvoření přijatelného barevného schématu nebo vytvoření ikon u tlačítek, pro přijatelnější ovládání.

Základní rozvržení panelů jsem sice převzal z původní verze programu, ale bylo třeba ještě provést jeho dodatečné úpravy. Pro tento účel jsem použil plugin pro Eclipse s názvem WindowBuilder, který mi v této oblasti usnadnil práci. Následný kód jsem poté ale ještě dodatečně ručně upravoval. Kromě rozvržení jednotlivých grafických komponent bylo ještě třeba vytvořit ikony pro lepší práci v aplikaci. Ikony pro tlačítka, sloužící k navrácení předchozího stavu nebo pro sloučení

objektů, byly získány z volně přístupné databáze ikon, kterou lze nalézt na [2]. Ikony reprezentující jednotlivé typy tvarů jsem však již vytvořil sám. K tomu účelu jsem využil možnosti opensource vektorového grafického editoru s názvem Inkscape. Porovnání původního a výsledného vzhledu můžete vidět na následujících obrázcích.




*Obrázek 9 .1: Vzhled původní aplikace*




*Obrázek 9 .2: Výsledný vzhled aplikace*

# 10 Závěr

V této bakalářské práci jsem rozšířil stávající prototyp aplikace pro práci s vektorovou grafikou. Hlavním účelem rozšíření bylo uživateli poskytnout větší množství typů vkládaných objektů. U všech takto implementovaných tvarů byly přidány možnosti jejich dodatečné editace, či jejich seskupování do větších celků. Nešlo zde však jen o pouhé doimplementování nových objektů, ale vzhledem k nedostačujícímu předchozímu zpracování bylo třeba upravit velkou část programu a změnit celý systém vykreslování objektů.

Aplikaci bylo dále přepracováno grafické uživatelské rozhraní, které neodpovídalo představám uživatele. Byl zde například přidán panel zobrazující seznam všech vytvořených objektů zohledňující i jejich hierarchické uspořádání, dále byl vytvořen postranní panel, dík kterému může uživatel zjišťovat parametry označeného objektu a následně je také editovat. Mimo to bylo ještě pozměněno barevné schéma celé aplikace a upraveny byly také základní tlačítka a celý seznam obsahující všechny dostupné typy tvarů.

Tato práce mi nepřinesla pouze zdokonalení v jazyce Java, ale přinesla mi i první zkušenosti pro práci v týmu na společném projektu. Souběžně s touto prací soustřeďující se na grafické rozšíření vytvořeného vektorového animátoru bylo vypsáno i zadání pro rozšíření jeho animační části. Z důvodu práce na stejné aplikaci bylo proto třeba dodržet úzkou spolupráci mezi těmito dvěma projekty. Pro tyto účely bylo například dohodnuto použití verzovacího systému TortoiseSVN, díky kterému bylo zajištěno lepší sdílení celého kódu. Spolupráce byla nutná i v samotné implementaci, kdy se například bylo třeba dohodnout na rozhraní, přes které budou objekty komunikovat s animacemi.

# Použitá literatura

[1] *Oracle documentation* [online]. [cit. 2013-04-30]. Dostupné z: http://docs.oracle.com/

[2] *FindIcons* [online]. [cit. 2013-04-30]. Dostupné z: http://findicons.com/

[3] *ApacheCommons: Commons BeanUtils* [online]. [cit. 2013-04-30]. Dostupné z: http://commons.apache.org/

[4] Full-justification with a Java Graphics.drawString replacement. *StackOverflow* [online]. 2008 [cit. 2013-04-30]. Dostupné z: http://stackoverflow.com/questions/400566/full-justification-with-a-java-graphics-drawstring-replacement

[5] JTree set node name as one of UserObject attribute. *StackOverflow* [online]. 2012 [cit. 2013-04-30]. Dostupné z: http://stackoverflow.com/questions/9428538/jtree-set-node-name-as-one-of-userobject-attribute

[6] PECINOVSKÝ, Rudolf. *Návrhové vzory*. Vyd. 1. Brno: Computer Press, 2007, 527 s. ISBN 978-80-251-1582-4.

[7] ŠUTA, Karel. *Vektorový animátor*. Ostrava, 2009. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky. Vedoucí práce David Ježek.

# Seznam příloh



Součástí BP je CD.

Adresářová struktura přiloženého CD:

| Adresář | Obsah |
|---|---|
| /Text/ | Text bakalářské práce |
| /Prirucka/ | Uživatelská dokumentace |
| /Aplikace/ | Zdrojový kód aplikace |
| /Ukazka/ | Předpřipravené animační scény |

# Uživatelská dokumentace

Zpracoval: Martin Gold, Zděnek Gold

## Úvod

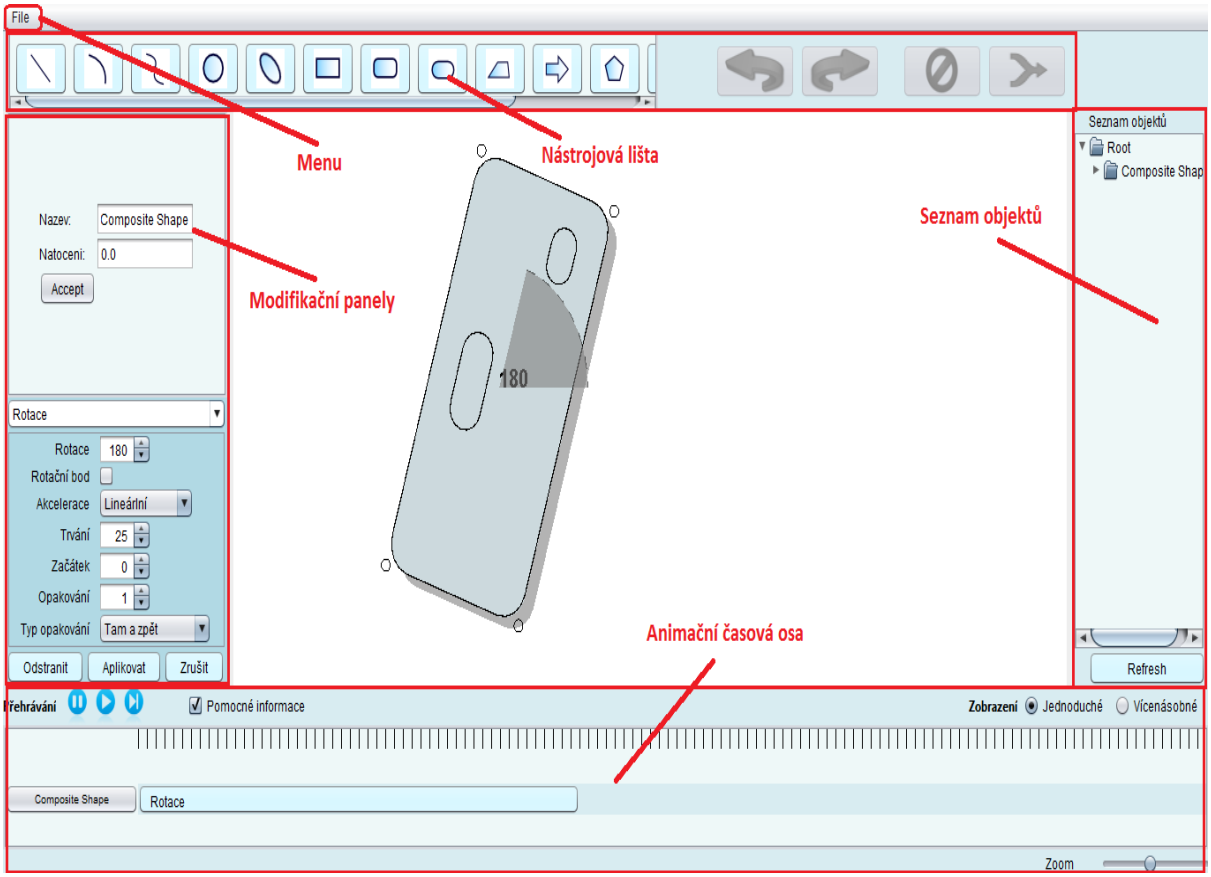
Tento grafický animátor byl vytvořen jako bakalářská práce Martina Golda a Zdeňka Golda. Jde o jednoduchou aplikaci, která by měla být vhodná pro tvorbu materiálů pro výukové účely.

Program umožňuje vytváření objektů za pomoci již nadefinovaných tvarů včetně možnosti jejich následného seskupování do větších celků. Ke každému objektu zde existuje i výčet parametrů, pomocí kterých lze tvary editovat.

K vytvořenému objektu lze následně přiřadit jednu nebo více předdefinovaných animací, jejichž nastavení lze taktéž ovlivňovat.

Výslednou animaci lze nakonec vyexportovat do video souboru typu MPEG4.

# Grafické uživatelské rozhraní aplikace




# Menu

Hlavní panel obsahuje přístup k možnostem týkajících se celého projektu. Obsaženo je zde jen jedno tlačítko, nazvané Soubor, pod kterým se nacházejí tyto možnosti:

- Uložení animace – umožňuje uložení celého projektu včetně vytvořených animací
- Načtení animace – umožňuje načíst již dříve vytvořený projekt
- Uložení tvaru – umožňuje uložit nadefinovaný tvar
- Načtení tvaru – umožňuje načíst již dříve vytvořený tvar
- Vytvoř video – umožňuje vyexportovat vypracovanou animaci do spustitelného video souboru MPEG4

# Nástrojová lišta

Nástrojová lišta tvoří jednu z nejpodstatnějších částí aplikace. Jsou zde umístěny jak základní tlačítka pro ovládání programu, tak i seznam objektů, které lze vytvářet. Mezi základní tlačítka pro ovládání patří možnost postoupit dopředu/zpět, které umožňují posunovat dále/zpět stav pomocí záchytných bodů uložených v historii změn. Dále se zde nachází ještě tlačítko pro odstranění objektu

(pozn. u sloučených objektů slouží pouze k odstranění celkové vazby a zachovává tak všechny členské objekty) a tlačítko pro sloučení více objektů do jedné skupiny. Nakonec je zde ještě umístěn seznam tlačítek, pomocí kterých lze vytvářet všechny předdefinované tvary.

Seznam obsahuje (zleva): úsečku (line), Bézierovu kvadratickou křivku (Bezier quadratic curve), Bézierovu kubickou křivku (Bézier cubic curve), kruh (circle), elipsu (ellipse), obdélník (rectangle), obdélník s kulatými rohy (rounded rectangle), ovál (oval), lichoběžník (trapezoid), šipku (arow), n-úhelník (ngon), hvězdu (star), kvádr (block) a textové pole (text box).

# Modifikační panely

Modifikační panely se nacházejí na levém okraji aplikace a umožnují uživateli editovat jak strukturu objektu, tak i průběh jeho animace.

### Editace objektu

V této části aplikace jsou uživateli nabídnuty veškeré parametry, pomocí kterých lze upravovat celkový vzhled právě označeného objektu. Tyto parametry nabízí jak základní modifikace, které se týkají například výšky a šířky tvaru, tak i pokročilejší funkce. Mezi ně patří například možnost zapnout či vypnout stín, měnit odsazení stínu, měnit barvy výplně a ohraničení, měnit styl ohraničení nebo například modifikovat celý objekt pomocí transformace zkosením. Seznam těchto funkcí však zde nemůže být vypsán uceleně, poněvadž výčet aktuálních funkcí záleží na právě označeném objektu. Některé tvary totiž nabízí jen malý výběr funkcí a naopak některé tvary nabízí ještě navíc některé speciální vlastnosti charakteristické pouze pro ně. Příkladem může být textové pole, které kromě běžných funkcí nabízí uživateli ještě široké možnosti práce s textem.

### Editace animací

V panelu pro editaci animací se uživateli nabízí prostředky k definování vlastností animací nebo k jejich přidávání a odebírání. Součástí tohoto panelu je i vysouvací seznam s nabídkou dostupných typů animací. U každého typu animace se stanovují společné parametry: snímek počátku animace, délka animace a akcelerátor. Některé speciální animace, kromě těchto základních vlastností využívají hodnoty v podobě vektoru, rotačního bodu apod. Také existují možnosti nastavit počet a typ opakování. V dolní části panelu jsou tři tlačítka:

- Přidat/Odstranit - přidává/odstraňuje aktuálně definovanou animaci do/ze scény
- Aplikovat - potvrzuje nastavené hodnoty nastavené v modifikačním panelu a aplikuje je na animaci
- Zrušit - zahazuje současně nastavené hodnoty aplikované na animaci

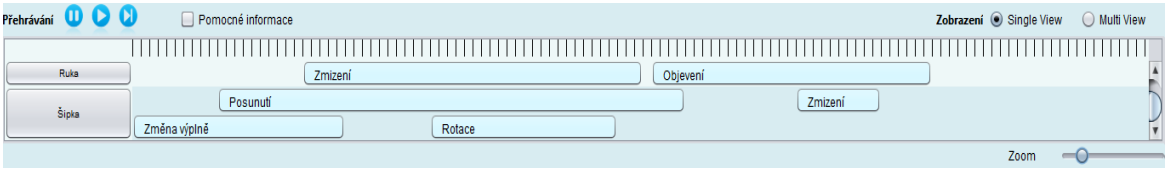
# Seznam objektů

Tento panel je umístěn na pravém okraji aplikace a slouží k snadnější orientaci v programu. Hlavní částí je okno, ve kterém se vypisuje seznam jmen všech použitých objektů v tomto projektu. Seznam však uživateli nenabízí prostý výčet jmen, ale dbá také na zachovávání jejich hierarchické struktury. Ta je využívána hlavně při shlukování objektů do skupin. V tomto případě se v seznamu vytvoří adresář reprezentující takto sloučenou skupinu a do něj jsou přesunuty všechny členské

objekty. S tímto seznamem je pak spojeno tlačítko Aktualizovat, které po jeho stisknutí přepracuje celý zobrazený seznam.

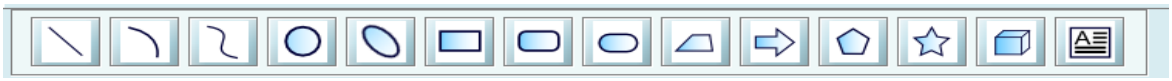
## Animační časová osa

V dolní části okna aplikace je umístěna komponenta pro výběr animací a nastavování aktuálního snímku scény. V levém horním rohu tohoto panelu jsou tlačítka k ovládání náhledu animování objektů. Zaškrtávací tlačítko povolí nebo zakáže vykreslování informací o animacích přímo na plátně. V pravém dolním rohu panelu je posuvný jezdec umožňující přibližování/oddalování časové osy.



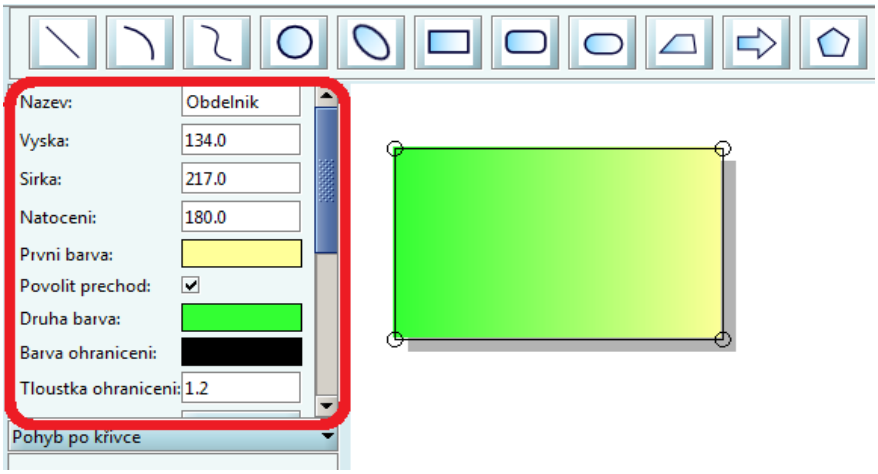

## Vytváření a editace objektů

Vytváření jednotlivých objektů je velmi jednoduché. Nejdříve si uživatel vybere jeden z předdefinovaných tvarů nacházejících se v seznamu objektů. Vybraný objekt poté vytvoříme kliknutím na kreslící plátno a následným tahem myši. Pro přehlednější vytváření tvarů je uživateli průběžně během tahu zobrazována aktuální velikost objektu.

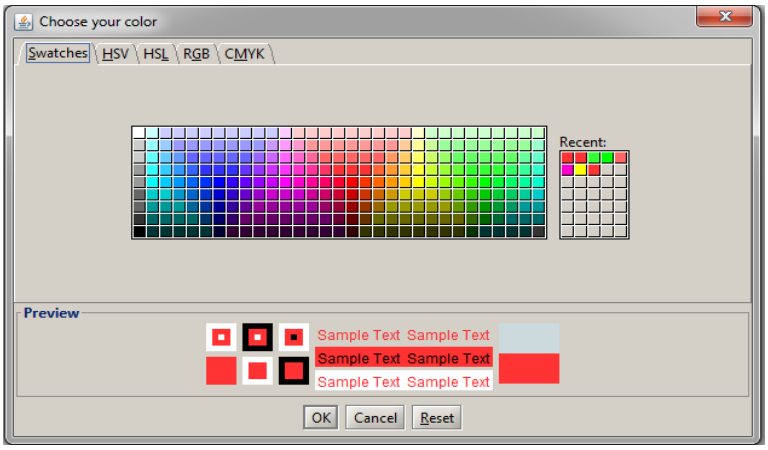


Pro označování objektů je možno použít jak kliknutí na daný tvar, tak možnost obdélníkového výběru, pomocí něhož lze vybrat i více objektů najednou. Označenému objektu se následně pro lepší přehlednost zvýrazní všechny jeho řídící body.

Kliknutím a následným tahem lze takto označený objekt přesouvat. Objekt však můžeme i otočit. To provedeme tak, že označíme jeden z řídících bodů tvaru, ten bude představovat střed otáčení. Poté opět využijeme funkce kliku a tahu, pomocí kterého určíme úhel otočení tohoto objektu.



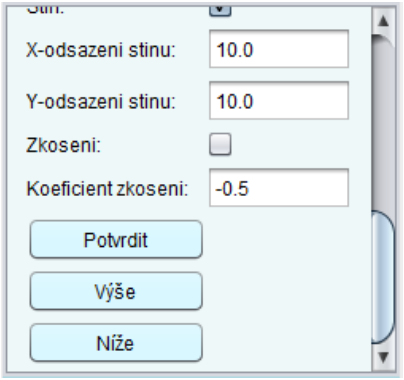

Po označení objektu lze také pozorovat překreslení modifikačního panelu, nacházejícího se na levé straně aplikace. Jestliže označíme více než jeden objekt nebo naopak žádný, zobrazí se uživateli prázdný dialog. Při označení právě jednoho objektu se však v tomto panelu zobrazí výčet všech parametrů, které můžou být u tohoto typu tvaru měněny.




V tomto okně pak může uživatel měnit všechny dostupné parametry a to buď přepsáním jejich hodnoty, zaškrtnutím nabízené možnosti nebo jednoduchým vybráním jedné možnosti z předpřipravené nabídky. Mimo to se zde vyskytují i atribut pro změnu barvy. Jejich změna se provádí klinutím na pole s barvou. Po této akci se uživateli zobrazí nové okno, které nabízí širokou škálu možností pro nadefinování požadované barvy. Může být využito například barevné schéma RGB, HSV, HSL nebo CMYK, kde lze nadefinovat i průhlednost určené barvy. Kromě těchto barevných schémat však může být využita i paleta s předdefinovanými barvami.

Změny, které nijak neovlivňují geometrii objektu (jako je například změna barvy) se projeví okamžitě. U ostatních změn je třeba potvrdit jejich nové hodnoty pomocí potvrzovacího tlačítka, které

lze vidět na obrázku níže. Povšimnout si zde můžeme i dalších dvou tlačítek, při jejichž stisknutí se označený objekt přesune buďto více do pozadí nebo naopak více do popředí.

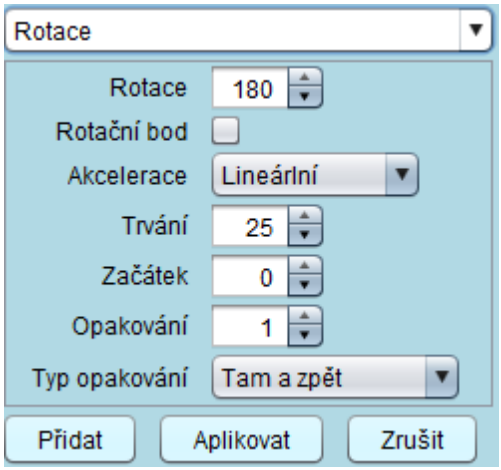


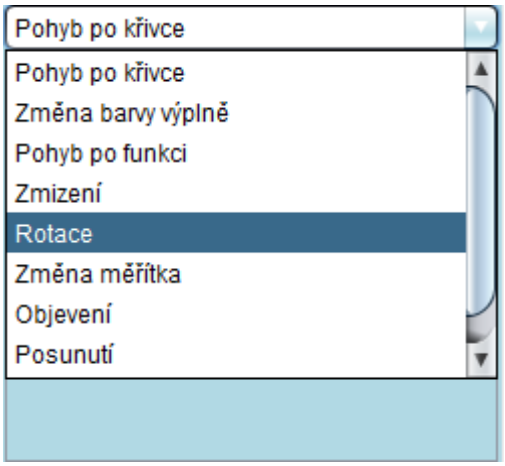

# Přiřazování a editace animací

Chceme-li s vytvořenými objekty vytvořit animovanou scénu, je třeba zvolený objekt nejprve označit. Po výběru objektu se přesuneme do modifikačního panelu a z vysouvací nabídky si vybereme typ animace. V základní verzi programu máme k dispozici tyto typy animací:

- Pohyb po křivce - objekt opisuje trajektorii kubické bézierové křivky
- Změna barvy výplně - objekt plynulým přechodem mění barvu výplně na zvolenou barvu
- Pohyb po funkci - objekt opisuje trajektorii definovanou nějakou funkcí
- Zmizení - objekt mění svoji průhlednost do ztracena
- Rotace - objekt se otáčí kolem svého středu nebo kolem bodu mimo střed
- Změna měřítka - objekt plynule mění výšku a šířku podle násobku těchto velikostí
- Objevení - objekt mění svoji průhlednost až do plného zobrazení
- Posunutí - objekt se přesune o stanovený vektor posunutí
- Animace průhlednosti - objekt mění svoji průhlednost do nastavené hodnoty průhlednosti

Po zvolení typu animace se v editačním panelu zobrazí její parametry. U každého objektu se nastavují různé parametry. Viz obrázek níže.

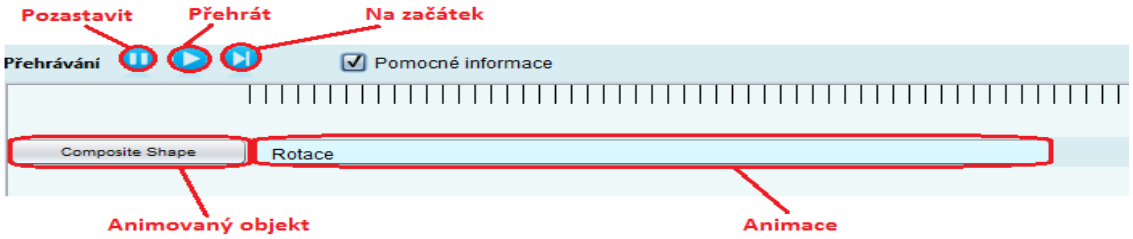


Nyní je na nás, jak nastavíme řídící hodnoty animace. U každé animace budou vždy vlastnosti:

- Akcelerace - nastavuje zrychlení animace
- Trvání - nastavuje délku animace ve snímcích
- Začátek - snímek, ve kterém animace začíná
- Opakování - počet opakovacích cyklů
- Typ opakování - k dispozici jsou dvě možnosti (Tam a zpět - po dokončení jednoho cyklu se změní směr a animuje se pozpátku, Vždy od začátku - po dokončení cyklu se animace vrací do počátečního stavu a další cyklus probíhá od začátku)

Po nastavení všech hodnot, přidáme tlačítkem “Přidat” animaci do scény. Po přidání se animace objeví v časové ose. Tlačítkem “Zrušit” zahodíme všechny změny a animaci nepřidáme.




Nyní, když už je animace přidána do scény, můžeme si ji přehrát. K tomu slouží tlačítko na časové ose. Pokud zjistíme, že animace nám nevyhovuje, stačí na ní tlačítkem myši kliknout. Aktuální nastavení vybrané animace se nám znovu zobrazí v editačním panelu. Po změně hodnot stačí stisknout tlačítko “Aplikovat” a animace si uloží nové nastavení.

Pokud chceme animaci ze scény odebrat, musíme ji nejprve vybrat z časové osy a poté tlačítkem “Odstranit” ji v editačním panelu odstraníme.